PHP и MySQL

Исчерпывающее руководство



O'REILLY®

ПИТЕР®

Бретт Маклафлин

PHP and MySQL



the missing manual®

The book that should have been in the box®

Brett McLaughlin



Beijing | Cambridge | Farnham | Köln | Sebastopol | Tokyo

PHP и MySQL



Исчерпывающее руководство

Бретт Маклафлин



Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2013

ББК 32.988-02-018
УДК 004.738.5
М15

Маклафлин Б.

М15 PHP и MySQL. Исчерпывающее руководство. — СПб.: Питер, 2013. — 512 с.: ил.

ISBN 978-5-459-01550-8

Если у вас есть опыт разработки сайтов с помощью CSS и JavaScript, то эта книга переведет вас на новый уровень веб-разработки — создание динамических веб-сайтов на основе PHP и MySQL. С помощью практических примеров в книге вы узнаете все возможности серверного программирования. Вы прочитаете, как выстраивать базу данных, управлять контентом и обмениваться информацией с пользователями, применяя запросы и веб-формы.

— Написание PHP-скриптов и создание веб-форм.

— Синтаксис PHP и SQL.

— Создание и управление базой данных.

— Создание динамических веб-страниц, которые изменяются при каждом новом просмотре.

— Разработка шаблонов страниц об ошибках, которые будут выводиться пользователям.

— Использование файловой системы для доступа к данным пользователя, включая иллюстрации и двоичные файлы.

— Создание административной страницы для управления сайтом.

ББК 32.988-02-018
УДК 004.738.5

ISBN 978-0596515867 англ.
ISBN 978-5-459-01550-8

Краткое содержание

Введение 14
Об авторе. 22
От издательства 24

Часть 1. Основы PHP и MySQL

Глава 1. PHP: что, где и зачем? 26
Глава 2. Синтаксис PHP: удивительный и таинственный 50
Глава 3. MySQL и SQL: база данных и язык 82

Часть 2. Динамические веб-страницы

Глава 4. Подключение PHP к MySQL 120
Глава 5. Улучшение поиска с помощью регулярных выражений 157
Глава 6. Создание динамических веб-страниц. 174

Часть 3. Переход от веб-страниц к веб-приложениям

Глава 7. Когда что-то не получается (но должно получаться) 222
Глава 8. Обработка изображений и решение более сложных задач . . .260
Глава 9. Двоичные объекты и загрузка изображений. 294
Глава 10. Вывод списков, итерация и администрирование 337

Часть 4. Безопасность и реальное окружение

Глава 11. Аутентификация и авторизация 392
Глава 12. Cookie-файлы, вопросы регистрации и избавление от примитивных окон .428
Глава 13. Авторизация и сессии. 469

Оглавление

Введение 14
Что такое PHP? 14
PHP ориентирован на работу в сети 15
PHP не дает столько вольностей, сколько допускается в JavaScript . . . 17
PHP — это интерпретатор 17
Что такое MySQL? 18
Об этой книге 18
Macintosh и Windows 19
FTP: важная деталь 19
Структура книги 19
Внешние ресурсы 20
Недостающий компакт-диск 20
Отзывы 21

Об авторе 22
О творческой группе 22
Благодарности 23

От издательства 24

Часть 1. Основы PHP и MySQL

Глава 1. PHP: что, где и зачем? 26
Подбор инструментов 26
PHP на персональном компьютере (PC) 27
PHP на компьютерах Macintosh 32
Подбор текстового редактора 36

Создание вашей первой программы 39
Запуск вашей первой программы 41
Создание вашей второй программы 42
Начало работы: создание HTML-страницы 42
Создание PHP-сценария 43
Изменчивость переменных. 45
Выкладывание кода HTML, CSS и PHP 46
Запуск вашей второй программы 48

Глава 2. Синтаксис PHP: удивительный и таинственный 50
Получение информации из веб-формы 50
Непосредственный доступ к параметрам запроса 50
Создание собственных переменных 54
Работа с текстом в PHP 58
Объединение текста 58
Поиск в тексте. 60
Изменение текста 65
Обрезка и замена текста 68
Переменная $_REQUEST 73
Массивы могут содержать несколько значений. 74
Работа с $_REQUEST как с массивом 76
Что делать с пользовательской информацией?. 80

Глава 3. MySQL и SQL: база данных и язык 82
Что такое база данных?. 82
Базы данных являются постоянным хранилищем 82
База данных — это в первую очередь структура. 84
(Хорошие) базы данных являются реляционными. 86
Установка MySQL. 86
MySQL для Windows 88
MySQL для Mac OS X 94
Запуск вашего первого SQL-запроса 100
SQL — язык для разговора с базами данных .105
Вход в базу данных вашего веб-сервера. 107
Использование базы данных с помощью команды USE 109
Создание таблиц с помощью инструкции CREATE 110
Удаление таблиц с помощью команды DROP 114
Вставка нескольких строк с помощью команды INSERT. 115
И в завершение — команда SELECT 116

Часть 2. Динамические веб-страницы

Глава 4. Подключение PHP к MySQL 120
Создание простого PHP-сценария, предназначенного для подключения 120
Подключение к базе данных MySQL 121
Выбор используемой базы данных 125
Демонстрация таблиц базы данных с помощью команды SHOW 127
Обработка ошибок путем наблюдения за отсутствием результата. . . .128
Вывод SQL-результатов 129
Приведение кода в порядок с помощью нескольких файлов. 133
Замена набранных вручную значений переменными 134
Абстрагирование важных значений путем заключения их в отдельный файл 135
Переменные изменяются, а константы сохраняют постоянство. 137
Создание элементарного исполнителя SQL-запросов 140
Создание HTML-формы с большим пустым полем ввода 140
Подключение к базе данных (еще раз). 142
Запуск пользовательского SQL-запроса (еще раз). 143
Ввод вашего первого запроса, основанного на применении веб-технологий. 145
Обработка запросов, не выбирающих информацию с помощью команды SELECT 148
Учет человеческого фактора 152
По возможности нужно избегать внесения изменений в пользовательский ввод 153

Глава 5. Улучшение поиска с помощью регулярных выражений 157
Сопоставление строк, двойная скорость 158
Простая программа поиска в строке. 158
Поиск одной строки... или другой. 160
Учет позиции. 164
Избавление от trim и strtoupper 166
Поиск набора символов. 168
Регулярные выражения: к бесконечности и еще дальше. 172

Глава 6. Создание динамических веб-страниц. 174
Повторное обращение к пользовательской информации 174
Проектирование таблиц базы данных 176
В правильных таблицах баз данных всегда есть столбец id. 177
Ваш друг автоприращение. 178

ID и первичные ключи — хорошие компаньоны 178
Добавление ограничений к базе данных 180
Сохранение информации о пользователе. 182
Создание SQL-запроса 183
Вставка данных о пользователе 185
Первое действие по подтверждению 188
Не нужно путать пользователей с программистами 189
Покажите мне пользователя 191
Макетирование страницы профиля пользователя 191
Изменение структуры таблицы с помощью ALTER 194
Создание сценария: первый проход 196
Выбор пользователя из базы данных с помощью инструкции SELECT 201
Извлечение значений из результата SQL-запроса 204
Получение ID пользователя сценарием show_user.php 207
Перенаправление и повторное обращение к сценарию, создающему новых пользователей 210
Обновление формы регистрации 210
Обновление сценария создания пользователя .213
Усовершенствование кода с помощью регулярных выражений (в очередной раз). 216

Часть 3. Переход от веб-страниц к веб-приложениям

Глава 7. Когда что-то не получается (но должно получаться) 222
Проектирование страниц ошибок. 224
Что должны видеть пользователи?. 225
Понятие о том, когда и сколько нужно говорить. 228
Поиск компромисса для страниц ошибок с помощью PHP 230
Создание страницы ошибки с кодом PHP 231
Проверка принятого решения 233
Ожидайте неожиданного 234
А теперь вас ждут проблемы безопасности и фишинга 237
Добавление отладки к приложению. 240
Включение отчета об ошибках, выдаваемого интерпретатором PHP 241

Переход от require к require_once 247
Сейчас вы меня видите, а сейчас нет 249
Переадресация на ошибку 251
Обновление вашего сценария для использования show_error.php 251
Простота и абстракция 254
Переадресация не видит пути к файлу 256

Глава 8. Обработка изображений и решение более сложных задач 260
Изображения — это просто файлы. 261
Формы HTML могут готовить почву 263
Отправка изображения пользователя на ваш сервер 266
Были ли ошибки при отправке файла? 270
Сохранение местоположения изображения в базе данных 279
Изображения, предназначенные для просмотра. 282
Выбор изображения с помощью инструкции SELECT и вывод его на экран 283
Преобразование путей файловой системы в URL-адреса. 285
Отображение картинки вашего пользователя: дубль два 289
Несколько небольших исправлений в app_config.php 291
А теперь совсем о другом 293

Глава 9. Двоичные объекты и загрузка изображений. 294
Хранение разных объектов в различных таблицах 294
Вставка в таблицу необработанного изображения 297
Функция getimagesize не возвращает размер файла 300
Функция file_get_contents оправдывает свое название 300
Вставка изображения с помощью инструкции INSERT. 300
Пока ваши двоичные данные вставлять небезопасно 301
Вставка строки в переменную 302
Получение правильного ID перед перенаправлением. 305
Связывание пользователей и изображений 308
Вставка изображения при вставке пользователя 310
Связывание таблиц с помощью условия WHERE 316
Покажите мне изображение. 318
Вывод изображения. 319
Перехват и обработка ошибок 324
Тест, тест и еще раз тест 328

Встроить изображение ничуть не сложнее, чем его просмотреть 329
Вам нужен лишь идентификатор изображения 329
Сценарий может быть в виде указания в теге источника изображения (src) 330
Итак, какой же подход лучше?. 335
Ладно, если вы непременно хотите получить конкретный ответ... . . .335

Глава 10. Вывод списков, итерация и администрирование 337
Вещи, которые никогда не меняются 338
Пользовательский интерфейс, или Краткость по-прежнему сестра таланта 338
Нужен также список пожеланий. 340
Вывод списка всех пользователей 341
Выбор с помощью SELECT нужной (на данный момент) информации 342
Создание простой страницы администрирования 344
Перебор элементов массива. 346
Удаление пользователя 350
Разбор отдельных компонентов 350
Объединение всех составляющих. 351
Удаление пользователей не должно быть некой тайной операцией. 354
Возражения, высказываемые вашим пользователям. 358
У перенаправления есть некоторые ограничения 359
Возвращение окна предупреждения, создаваемого с помощью JavaScript 362
Функция alert прерывает действия 368
Приведение сообщений к единому стандарту. 369
Создание новой сервисной функции для отображения 372
Появление дубликатов — вполне ожидаемая проблема 374
Коды сценариев View и Display имеют общий характер. 376
Интеграция утилит, представлений и сообщений. 377
Вызов повторяющегося кода из сценария View. .377
Лучше использовать гибкие функции. 378
Стандартизация и объединение вывода сообщений в сценарии View 384
Создание функции для вызова двух функций 386
Теперь нужно просто распространить эту информацию на весь код. 387

Часть 4. Безопасность и реальное окружение

Глава 11. Аутентификация и авторизация 392
Начнем со стандартной аутентификации 393
Стандартная аутентификация с использованием HTTP-заголовков . . .394
Стандартная аутентификация проводится... стандартно 395
Самая худшая из всех аутентификаций 396
Получение данных о полномочиях вашего пользователя 397
Отмена не подходит для аутентификации 398
Получение пользовательских полномочий (на этот раз всерьез!) 399
Извлечение всего одинакового 403
Еще один сервисный сценарий: authorize.php 403
Пароли не должны находиться в сценариях PHP .407
Обновление таблицы users 408
Работа с вновь созданными недопустимыми данными. 409
Вам нужно получить исходные имя пользователя и пароль 411
Вставка имени пользователя и пароля 415
Подключение сценария authorize.php к таблице users. 419
Пароли обеспечивают безопасность, но и сами они должны быть защищены 422
Шифрование текста с помощью функции crypt 423
Однонаправленное шифрование с помощью функции crypt 424
При шифровании используется соль 425

Глава 12. Cookie-файлы, вопросы регистрации и избавление от примитивных окон. 428
Выход за рамки стандартной аутентификации 429
Начало с создания стартовой страницы 430
Управление регистрацией пользователя при входе в приложение . . .432
От HTTP-аутентификации к использованию cookie-файлов 435
Регистрация при входе в приложение с использованием cookie-файлов 436
Зарегистрировался пользователь или нет?. 438
Пытается ли пользователь зарегистрироваться?. 438
Отображение страницы 440
Перенаправление по мере необходимости 442
Регистрация пользователя при входе в приложение. 444
Пустые страницы и истечение срока действия cookie-файлов 447
Ошибки не всегда должны прерывать работу приложения 450
Настройка на повторные попытки 454

Добавление контекстно-зависимых меню. 455
Установка меню. 456
От HTML к сценариям 459
Отмена регистрации пользователей. 463
Требование создания cookie-файла 465

Глава 13. Авторизация и сессии. 469
Моделирование групп в базе данных. 469
Добавление таблицы groups 470
Отношение «многие ко многим» 471
Проведение теста на принадлежность к группе 475
Проверка на принадлежность к группе 476
Сценарий authorize.php нуждается в функции 477
Получение списка групп 479
Последовательный перебор групп 481
Разрешить, отказать, перенаправить 485
Меню, ориентированное на принадлежность к той или иной группе. 487
Введение в практику использования сессий браузера 491
Сессии находятся на серверной стороне 494
Сессии должны быть запущены 494
От $_COOKIE к $_SESSION. 495
Сессии должны быть еще и перезапущены. .496
Переменная $_REQUEST не включает в себя данные переменной $_SESSION. 500
Меню для любого пользователя?. 502
А теперь отмените регистрацию. 503
А вы не забыли о проблеме фишинга?. 504
А зачем вообще использовать cookie-файлы?. 507

Введение

Вы создали веб-страницу в HTML. Вы даже придали ей стилевое оформление с помощью каскадных таблиц стилей — Cascading Style Sheets (CSS) и написали небольшой код JavaScript для проверки допустимости данных, введенных в созданные вами веб-формы. Но этого было недостаточно, поэтому вы углубились в изучение JavaScript, включили в код библиотеку jQuery и сконструировали множество веб-страниц. Вы даже переместили свой код JavaScript во внешние файлы, сделали таблицы CSS общими для всего сайта и привели HTML в соответствие последним стандартам.

Но теперь вам захотелось большего.

Возможно, вас стало не устраивать, что ваш сайт не в состоянии хранить пользовательскую информацию в чем-нибудь, кроме cookie-файлов. Возможно, вам захотелось получить полнофункциональный интернет-магазин, поддерживающий оплату с помощью PayPal и с подробным списком того, что имеется в наличии. А возможно, вы просто обнаружили ошибку в программе и захотели выйти за рамки того, что вам могут дать HTML, CSS и JavaScript.

В любом случае изучение PHP и MySQL станет отличным способом сделать гигантский шаг вперед в программировании. Даже если вы никогда не слышали о PHP, вы поймете, что это лучшая возможность перейти от создания веб-страниц к разработке полноценных веб-приложений, хранящих всевозможную информацию в базах данных. Как это делается, показано в данной книге.

Что такое PHP?

PHP — это язык программирования. Он похож на JavaScript тем, что основную часть времени программист проводит за работой со значениями и принимает решения, по какому пути в коде нужно проследовать в каждый отдельно взятый момент времени. На HTML он похож тем, что программист работает с выводом — с тегами, которые посетители сайта просматривают через призму своих веб-браузеров. По сути, PHP в контексте веб-программирования проявляет свой универсальный характер: он неплохо справляется с решением многих вопросов, вместо того чтобы заниматься одним конкретным делом.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Что означает PHP?

PHP — это акроним. Изначально PHP расшифровывалось как Personal Home Page (персональная домашняя страница), поскольку множество программистов использовало его для создания своих веб-сайтов, применяя его гораздо чаще по сравнению с HTML, CSS и JavaScript. Однако в последние несколько лет фразой Personal Home Page можно, скорее, описать какой-то сайт, размещенный на дешевом хостинге, а не высокоэффективный язык программирования.

Поэтому теперь PHP означает PHP: Hypertext Preprocessor (PHP: гипертекстовый препроцессор). Возможно, это звучит несколько странно, из разряда программистских шуток: акроним PHP означает нечто, что само в себе содержит этот акроним. Этакий рекурсивный акроним, или в акроним, ссылающийся на самого себя. Вам не нужно знать, что такое рекурсивный акроним, поскольку здесь не викторина. Просто знайте, что рекурсивный акроним PHP не станет последней странной и забавной вещью, попавшейся вам в языке PHP.

PHP ориентирован на работу в сети

Если вы взяли в руки эту книгу с целью заняться веб-программированием, то вы на правильном пути. Хотя на PHP можно писать программы, запускаемые из командной строки (посмотрите на пример, показанный на рис. 0.1), истинная ценность PHP заключается не в этом.

Untitled

```
<?php

echo "Hello there. So I hear you're learning to be a PHP programmer!\n";
echo "Why don't you type in your name for me:\n";
$name = trim(fgets(STDIN));

echo "\nThanks, " . $name . ", it's really nice to meet you.\n\n";

?>
```

Рис. 0.1. PHP-программы можно запускать из окна терминала или из командной оболочки Windows

PHP уже готов к работе с HTML-формами, веб-сессиями и браузерными cookie-файлами. Он отлично встроится в существующую на вашем сайте систему аутентификации или же позволит вам создать свою собственную систему. Зачастую вы не только будете передавать управление HTML-странице, но и вписывать код HTML, с которым вы уже знакомы, непосредственно в код PHP. Вам часто

придется набирать код PHP, а затем код HTML, и все это, как показано в следующем примере, в одном и том же PHP-файле:

```
<?php

require '../../scripts/database_connection.php';

// Получение идентификатора пользователя, чьи данные нужно показать
$user_id = $_REQUEST['user_id'];

// Создание инструкции SELECT
$select_query = "SELECT * FROM users WHERE user_id = " . $user_id;

// Запуск запроса
$result = mysql_query($select_query);

// Присваивание значений переменным
?>

<html>
  <!-- Весь код HTML со вставками из кода PHP -->
</html>
```

Что получается в результате? Страницы, наполненные как HTML, так и динамическим контентом (рис. 0.2). На этой странице используется не меньше PHP, чем HTML. Она ищет имя посетителя в базе данных и отображает сведения о нем в динамическом режиме. В меню создается пункт Мой профиль, характерный для этого пользователя. Но здесь все еще очень много кода HTML. В этом PHP проявляется во всей своей красе: на странице уже известный вам HTML и даже JavaScript сочетаются с PHP, который вы собираетесь изучать.



Рис. 0.2. Страница с HTML и динамическим контентом

PHP не дает столько вольностей, сколько допускается в JavaScript

Если вы уже создавали программы на JavaScript, то вы уже знаете, что JavaScript позволяет делать практически все, что угодно. Вы можете случайно не поставить точку с запятой, можете использовать или не использовать скобки, можете применять или не применять ключевое слово `var`. Такая вольность, с одной стороны, ускоряет работу, а с другой — может принести и неприятности. Порой она затрудняет поиск ошибок и превращает работу в разных браузерах в сплошной кошмар.

PHP не допускает столько вольностей, сколько JavaScript, поэтому он требует дополнительного изучения структуры и понимания того, что происходит во время интерпретации программы. Это положительный момент, поскольку в конечном итоге вы подтянете и свое мастерство работы с JavaScript. Кроме того, строгое постоянство PHP упрощает его изучение. Вы получаете четкие правила, которых нужно придерживаться, а не массу фраз типа «Вы можете сделать это... или это... или это».

Итак, приготовьтесь. Вам нужно многое усвоить, но все изученное даст вам надежный фундамент для создания программ. И PHP сразу же сообщит, если возникнет какая-нибудь проблема. Вам не придется раскрывать консоль ошибки или отводить взгляд от небольшого желтого треугольника предупреждения, имеющегося в Internet Explorer, как это случалось при работе с JavaScript.

PHP — это интерпретатор

PHP-код существует в виде сценариев, которые являются простыми текстовыми файлами, написанными вами. Интерпретатор PHP представляет собой часть программного обеспечения (ПО) вашего веб-сервера, который читает этот файл, видит в нем определенный смысл, а затем возвращает веб-серверу HTML-вывод и направление дальнейших действий или порядок интерпретации записей пользовательской формы. Ваш текстовый файл интерпретируется построчно при каждом доступе к файлу.

Эта схема отличается от работы с такими языками, как Java или C++, которые компилируют код. При работе с этими языками вы пишете текстовые файлы, но затем запускаете команду, которая превращает эти текстовые файлы в нечто иное: файлы классов, двоичные файлы, части нечитаемого кода, используемого вашим компьютером.

Прелесть интерпретирующих языков, таких как PHP и JavaScript, состоит в том, что вы пишете код и тут же его запускаете. Вам не нужны пакеты инструментов или какие-то этапы обработки. Вы создаете код PHP. Тестируете его в браузере. Пишете дополнительный код. Таковы факты, и обычно это означает, что работа приносит удовольствие.

Что такое MySQL?

MySQL — это база данных. В ней хранятся ваша информация, данные ваших пользователей и другие нужные вам сведения. Наверняка, при знакомстве с MySQL и SQL (это язык, на котором осуществляется взаимодействие с MySQL) вы узнаете много новых нюансов. Подробнее с MySQL мы познакомимся в главе 3, когда у вас в запасе уже будут некоторые знания по PHP.

А сейчас считайте, что MySQL — это хранилище, куда можно класть вещи, которые позже можно будет там найти. Кроме того, MySQL предоставляет вам небольшого и действительно очень расторопного чертенка, который носится в поиске всего имущества, хранящегося на складе, как только оно понадобится. По мере чтения данной книги вам полюбится этот чертенок (простите)... MySQL. Он будет выполнять работу, с который вы сами бы никогда не справились, и будет делать это быстро и неутомимо.

Об этой книге

PHP является веб-языком, а не программой, поставляемой в коробке. Существуют десятки (если не сотни) тысяч веб-сайтов, располагающих PHP-инструкциями. Впечатляет, не так ли? Однако не все эти веб-сайты отвечают современным требованиям. Некоторые из них сильно критикуются другими программистами. Найти действительно подходящий сайт не так-то просто.

Цель этой книги — послужить руководством, которое должно быть под рукой уже в тот момент, когда вы загружаете PHP. На страницах издания вы найдете пошаговые инструкции для получения работоспособного PHP-интерпретатора, написания своей первой программы... и своей второй программы... и в конечном счете для создания с нуля целого веб-приложения. Кроме того, вы найдете четкий анализ самых важных компонентов PHP, которые будете ежедневно использовать при создании персонального блога или внутренней корпоративной сети.

Эта книга рассчитана на читателей с разным техническим уровнем. Основная часть материала предназначена для тех, кто уже прошел определенный путь от начала изучения веб-технологий, или для средних по уровню любителей веб-технологий и программистов. Я надеюсь, что вы уже разбираетесь в HTML и CSS и, может быть, даже немного знаете JavaScript. Если же все эти веб-компоненты вам незнакомы, специальные врезки под названием «К вашему сведению» предоставят вам вводную информацию, необходимую для понимания текущей темы. В свою очередь во врезках «Курсы повышения квалификации» предлагаются дополнительные технические советы, рассматриваются различные приемы и методы ускоренной работы для более-менее опытных любителей компьютеров.

Macintosh и Windows

Версии PHP и MySQL, предназначенные для Macintosh и Windows, работают почти одинаково. Самым важным является то, что основная часть работы проделывается за счет отправки ваших сценариев на веб-сервер и запуска на нем кода базы данных. Это означает, что вопросы операционной системы должен решать ваш хостинг-провайдер, а вам нужно сконцентрироваться на своем коде и информации.

Прочитав первые главы книги, вы получите систему, настроенную на работу с программным кодом и на работу с PHP-сценариями. Но вскоре вы забудете о том, под управлением какой операционной системы работаете: Macintosh или Windows. Вы просто будете создавать код. Точно так же, как пишете HTML и CSS.

FTP: важная деталь

Обратите внимание на то, что вам понадобится хорошая FTP-программа. Большинство PHP-программистов не сидят за удаленным сервером, набирая текст в редакторе командной строки вроде `vi` или `emacs`.

ПРИМЕЧАНИЕ АВТОРА

Набор текста в редакторе командной строки — одна из особенностей моей работы. Но я в некотором роде динозавр из тех минувших дней, когда приходилось смотреть коммерческие каналы, чтобы увидеть все интересные передачи, и когда терялись сообщения электронной почты, поскольку босс без всякого предупреждения отправлял свои команды по эфиру.

В наши дни для большинства из вас куда более подходящими будут хорошие текстовый редактор и графический FTP-клиент. А если серьезно, я уже давно хочу отказаться от своих причуд.

В главе 1 будут рассмотрены несколько хороших редакторов, причем некоторые из них имеют встроенный FTP-клиент. Но сейчас хотелось бы упомянуть программу Cyberduck (`www.cyberduck.ch`). Вы можете написать сценарий, отправить его по сети и все протестировать всего лишь несколькими щелчками кнопкой мыши. Итак, загрузите эту FTP-программу, настройте свой веб-сервер и приступайте к работе. Данная программа вам несомненно пригодится.

Структура книги

Эта книга состоит из четырех частей, каждая из которых содержит несколько глав.

- **Часть 1. Основы PHP и MySQL.** В первых трех главах данной части вы научитесь устанавливать PHP, запустите его на своем компьютере, напишете несколько своих первых программ на PHP и овладете некоторыми элементарными вещами, такими как сбор сведений о пользователе посредством веб-формы

и работа с текстом. Вы также установите MySQL и полностью ознакомитесь со структурой базы данных.

- **Часть 2. Динамические веб-страницы.** В этой части содержатся главы, с помощью которых вы приступите к созданию основ настоящего веб-приложения. Вы добавите таблицу, в которой сможете хранить сведения о пользователях, и получите представление о том, как легко вы можете манипулировать текстом. Для свободного манипулирования буквами, цифрами и слэшами в URL-адресах, адресах электронной почты, идентификаторах Twitter и т. д. будут использоваться регулярные выражения и управление строками.
- **Часть 3. Переход от веб-страниц к веб-приложениям.** Учитывая крепкую базу, полученную в предыдущих частях книги, вы уже будете готовы объединить свои веб-страницы в более цельный модуль. Вы добавите самостоятельно созданную систему обработки ошибок, чтобы ваши пользователи не пребывали в недоумении при сбое приложения, а также собственную отладку, помогающую обнаружить источники проблем. Кроме того, в базе данных будут сохранены ссылки на изображения, которые были предоставлены пользователями, а также сами изображения, и будет изучен вопрос, какой из этих двух подходов и в каких обстоятельствах будет предпочтительней.
- **Часть 4. Безопасность и реальное окружение.** Даже в самых простых приложениях регистрация при входе в приложение и отмена регистрации играют важную роль. В данной части вы создадите систему аутентификации, а затем займетесь паролями (которые при всей своей важности вызывают немало проблем). Затем мы рассмотрим работу с cookie-файлами и сессиями и использование этих технологий для создания в веб-приложении системы авторизации, основанной на принадлежности пользователя к той или иной группе.

На веб-сайте Missing Manual (www.missingmanuals.com/cds/phpmysqlmm) вы сможете найти все примеры кода из каждой главы.

Внешние ресурсы

Став владельцем данной книги, вы получили не только книгу для чтения. В Интернете вы найдете файлы примеров, позволяющие получить практический опыт, а также советы, статьи и, возможно, даже несколько видеороликов. Вы также можете связаться с командой, работающей над книгами этой серии, и сообщить нам о том, что вам понравилось (или не понравилось) в данном издании. Обратитесь на сайт www.missingmanuals.com или посетите описанные ниже разделы сайта.

Недостающий компакт-диск

У этой книги нет компакт-диска, приклеенного к обложке, но при этом вы ничего не теряете. Зайдите на сайт www.missingmanuals.com/cds/phpmysqlmm и скачайте примеры кодов. Там их предостаточно. Для каждой главы есть свой раздел кодов.

Кроме того, вам не придется сбивать пальцы на клавиатуре, набирая длинные веб-адреса. Упомянутая веб-страница предлагает также список ссылок на веб-сайты, приведенные в данном издании.

Отзывы

У вас есть вопросы? Нужна дополнительная информация? Видите себя в роли рецензента? На нашей странице отзывов вы можете получить ответы специалистов на вопросы, которые вам пришли в голову во время прочтения книги, поделиться мыслями о книгах данной серии и найти группу единомышленников, интересующихся PHP, MySQL и веб-приложениями в целом. Свяжитесь с нами, обратившись по адресу www.missingmanuals.com/feedback.

Об авторе

Бретт Маклафлин (Brett McLaughlin) — технолог и стратег высшего уровня, активно занимающийся веб-программированием и созданием управляемых данных, которые ориентированы на потребительские запросы систем. Редко сосредотачиваясь только на одном компоненте системы, он разрабатывает архитектуру и дизайн крупномасштабных приложений, руководит их созданием от начала и до конца с учетом жизненно важных параметров и контролем сроков реализации.

Разумеется, все это было рассказано, чтобы вы поняли, что Бретт является фанатом, проводящим основную часть своего времени за компьютером, и его руки просто летают по клавиатуре. В настоящее время он занимается в основном работой над проектами NASA, что звучит очень внушающе. Кто знает, может быть спутником, летящим сейчас над вами, действительно управляют с помощью PHP и MySQL...

О творческой группе

Нан Барбер (Nan Barber) — редактор, работает над серией Missing Manual со дня ее основания. Она живет в Бостоне со своим мужем в окружении различных электронных устройств. Адрес электронной почты: nanbarber@oreilly.com.

Жасмин Перес (Jasmine Perez) — возглавляет производственный отдел, в свободное время увлекается вегетарианской кулинарией, прослушиванием своей любимой радиостанции WFMU, вещающей в свободном формате, и походами. Адрес электронной почты: jperez@oreilly.com.

Нан Рейнхардт (Nan Reinhardt) — внештатный литературный редактор и корректор, который пишет к тому же романтические рассказы. В перерывах между редактированием она работает над своей третьей книгой. Три раза в неделю она ведет записи в своем блоге по адресу www.nanreinhardt.com. Адрес электронной почты: reinhardt8@comcast.net.

Шелли Поверс (Shelley Powers) — технический рецензент, входила в рабочую группу по HTML5 и была автором нескольких книг, вышедших в издательстве O'Reilly. Является также защитником животных. Веб-сайт: www.burningbird.net.

Стив Суэринг (Steve Suehring) — технический рецензент, разработчик с богатым опытом поиска простых решений сложных проблем. Стив играет на нескольких музыкальных инструментах (но не одновременно), с ним можно пообщаться на его веб-сайте www.braingia.org.

Благодарности

Благодарности практически никогда не удается выразить должным образом. Прежде чем говорить о самой книге, нужно выразить благодарность моей жене Ли, моим детям Дину, Робби и Адди. Любые радости или печали, случавшиеся в течение длительного процесса написания, делились с этой четверкой. Никакого авторского гонорара не хватит, чтобы компенсировать время, проведенное с ними. Я полагаю, что их терпение по отношению к моим писательским занятиям и оказываемая мне поддержка являются отражением их любви ко мне.

А затем, конечно, нужно перейти к благодарностям тем, кто работал над данным изданием. Брайан Сойер (Brian Sawyer) был первым, кто мне позвонил, когда у меня появилось время для писательской работы. И он позвонил мне в самый нужный момент: вдохновил на написание книги и вселил в меня уверенность, что я должен сделать это. А затем со мной связалась Нан Барбер (Nan Barber). Она оказала мне доверие, которое я не совсем заслужил. Я очень признателен ей, особенно за трудные дни начала августа, когда мне за несколько коротких недель оставалось написать еще сотни страниц.

Моими техническими рецензентами были Шелли Поверс (Shelley Powers) и Стив Суэринг (Steve Suehring), оба они были придирчивы, но учтивы. Шелли помогла мне не забыть, что перед собой всегда нужно видеть ученика. Если вам нравятся длинные листинги, в которых все начинает обрастать подробностями, то за это нужно благодарить ее. А Стив... Стив заполнил мои пробелы в PHP. Он решил одну особенно коварную проблему, что, как я думаю, существенно улучшило книгу. И если эта книга чем-то вам поможет, то ваша благодарность должна быть выражена и в его адрес.

А затем нужно упомянуть весь огромный механизм издательства O'Reilly. Он был задействован в полную силу. Я, правда, не знаю, как все это происходило, но я полностью удовлетворен их работой. Я представляю себе, как где-то Сандерс перемещает какие-то важные рычаги, Кортни изводит авторов, Лаура сердится и стучит каблуками, а Лори думает, что все это слишком много стоит, а Тим... думает о чем-то важном. Я всеми ими доволен.

Бретт Маклафлин

От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты halickaya@minsk.piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

ЧАСТЬ 1

Основы PHP и MySQL

Глава 1. PHP: что, где и зачем?
Глава 2. Синтаксис PHP: удивительный и таинственный
Глава 3. MySQL и SQL: база данных и язык

1 PHP: что, где и зачем?

PHP — это в конечном счете текст, получаемый веб-сервером и превращаемый в набор команд и информации для веб-браузера. Поскольку работа ведется с простым текстом, чтобы приступить к программированию на PHP, нужно будет всего лишь ознакомиться с самим языком PHP, и лучше всего этот сделать, установив PHP на свой компьютер, даже если большинство программ будет запускаться на веб-сервере.

Затем нужно будет запустить настоящий сценарий. Не волнуйтесь, создать свою первую программу на PHP совсем несложно, и до того, как приступить к изучению главы 2, вы создадите более одной программы.

Возможно, у вас возникнет вопрос: а зачем все это? Это нужно, чтобы все было под вашим контролем. PHP дает возможность активно принимать участие во всем, что делается на ваших веб-страницах. Вы сможете прислушиваться к запросам своих пользователей и грамотно на них отвечать. Поэтому приступайте к изучению PHP, не стоит больше оставлять своих пользователей с пассивными HTML-страницами.

Подбор инструментов

Прежде чем приступить к работе с PHP, нужно выполнить несколько действий. Создать веб-сайт без веб-браузера невозможно, и вы не можете написать PHP-программу без некоторых инструментов. Создание и настройка на компьютере собственной среды программирования на PHP не займет много времени.

Хотя PHP в отличие от веб-браузера не загружается заранее на каждый компьютер, этот язык можно легко загрузить из Интернета, заставить работать на вашем компьютере, и все это запустить, не потратив ни копейки. Кроме того, основная часть наипростейшего и лучшего инструментария для написания PHP-кода также находится в свободном доступе. Вам понадобится всего лишь собственная копия языка PHP, установленная на компьютере, плюс старый добрый текстовый редактор. Где их можно найти, будет показано в данном разделе.

PHP на персональном компьютере (PC)

Персональные компьютеры поставляются с большим количеством предустановленного программного обеспечения. К сожалению, большинство персональных компьютеров не комплектуется PHP. Но в этом нет ничего страшного: при наличии подключения к интернету PHP можно получить и запустить буквально за считанные минуты.

ПРИМЕЧАНИЕ

Если вы работаете на компьютере Macintosh, то этот процесс установки вам не придется выполнять. Переходите сразу к разделу «PHP на компьютерах Macintosh» данной главы.

Откройте свой рабочий веб-браузер и перейдите на веб-сайт www.php.net. Он является «интернет-домом» PHP, откуда вы и загрузите собственный экземпляр языка PHP, а также весь инструментарий, необходимый для написания и запуска PHP-программ. Найдите в правой части домашней страницы PHP заголовок стабильных выпусков — Stable Releases (рис. 1.1).

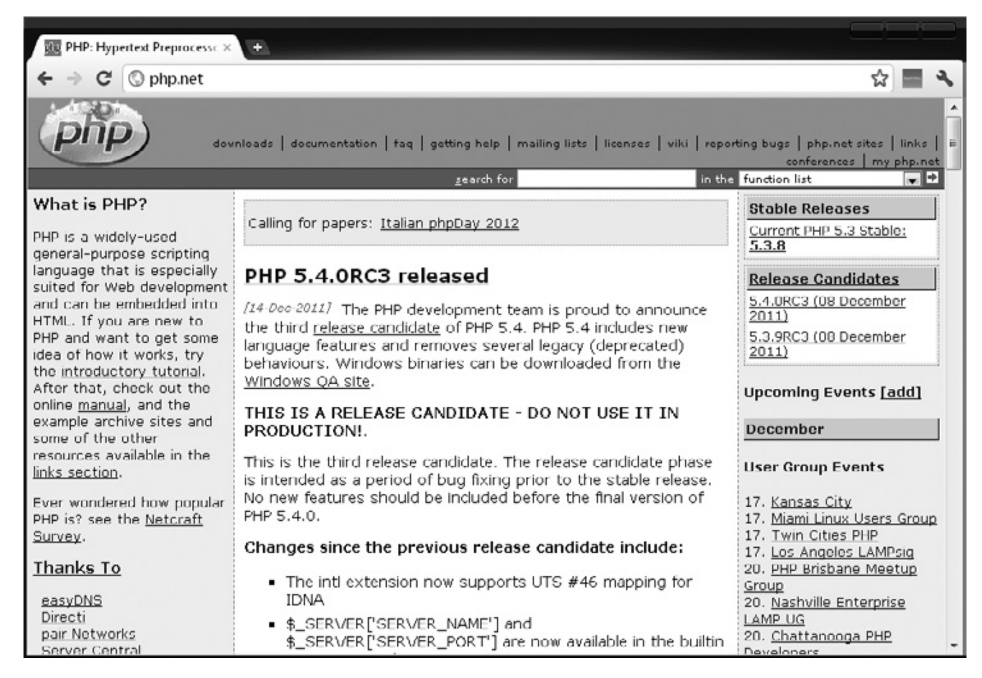


Рис. 1.1. При необходимости загрузить новую версию PHP или обновить имеющуюся посетите веб-сайт www.php.net

Щелкните на ссылке, указывающей на версию с наивысшим номером. (Дополнительные сведения о том, что означают эти версии, даны в ближайшей врезке.)

После выбора ссылки на версию PHP появится окно со ссылками на текущую версию PHP, а также как минимум на одну прежнюю версию (рис. 1.2). При отсутствии

каких-либо веских причин поступить иначе, всегда отдавайте предпочтение самой последней стабильно работающей версии.

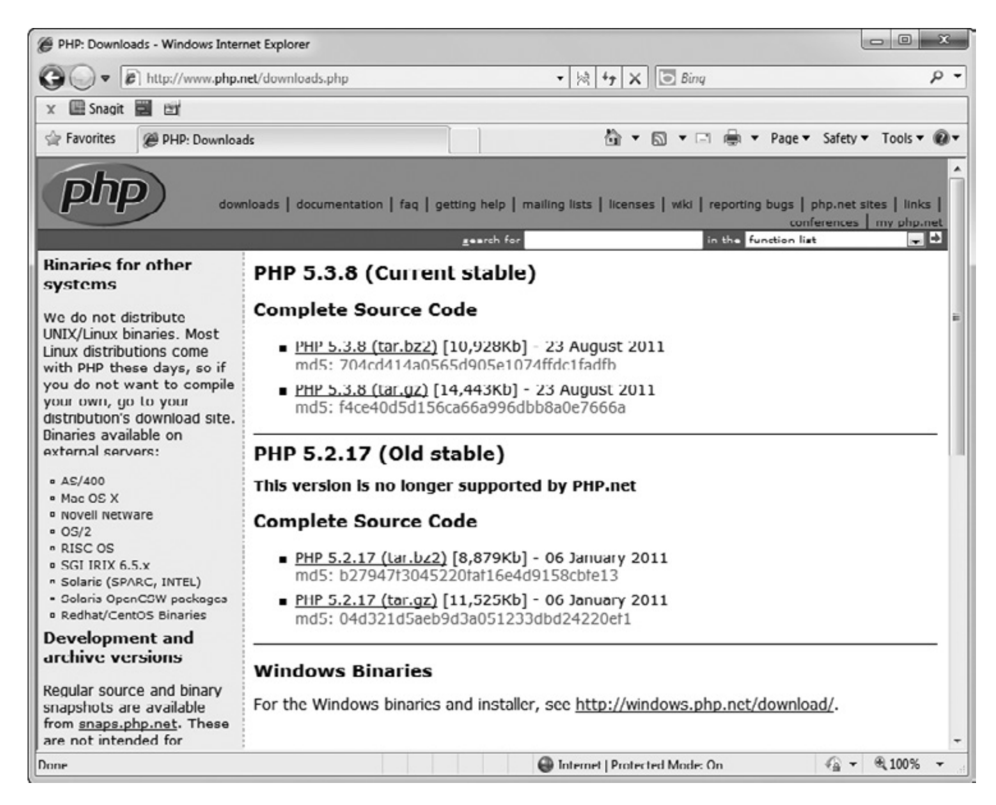


Рис. 1.2. На веб-сайте PHP всегда доступны для загрузки по крайней мере самая последняя стабильно работающая версия и предыдущая стабильно работающая версия

Перед загрузкой PHP посмотрите на нижнюю часть страницы с заголовком Windows Binaries (Двоичные коды для Windows). Это именно то, что вам нужно для получения PHP и его быстрого запуска на вашей Windows-машине. Щелчок на этой ссылке переправит вас на другой сайт: http://windows.php.net/download (рис. 1.3).

На странице http://windows.php.net/download есть варианты выбора самой последней версии и нескольких предыдущих версий. Для самой последней версии на странице будут два больших серых блока: первый для версии, небезопасной для потоков (Non Thread Safe), и второй для версии, безопасной для потоков (Thread Safe). Вам необходимо загрузить версию Non Thread Safe (Небезопасная для потоков), поскольку она работает намного быстрее. (Чем именно различаются друг от друга эти две версии сообщается в одной из следующих врезок.)

Просто найдите ссылку Installer (Установщик) и щелкните на ней. Установщик предполагает обычно длительную загрузку, но в него включен красивый Windows-установщик, существенно упрощающий запуск PHP. Щелкните на этой ссылке и выпейте чашку кофе, пока будете ждать завершения загрузки.

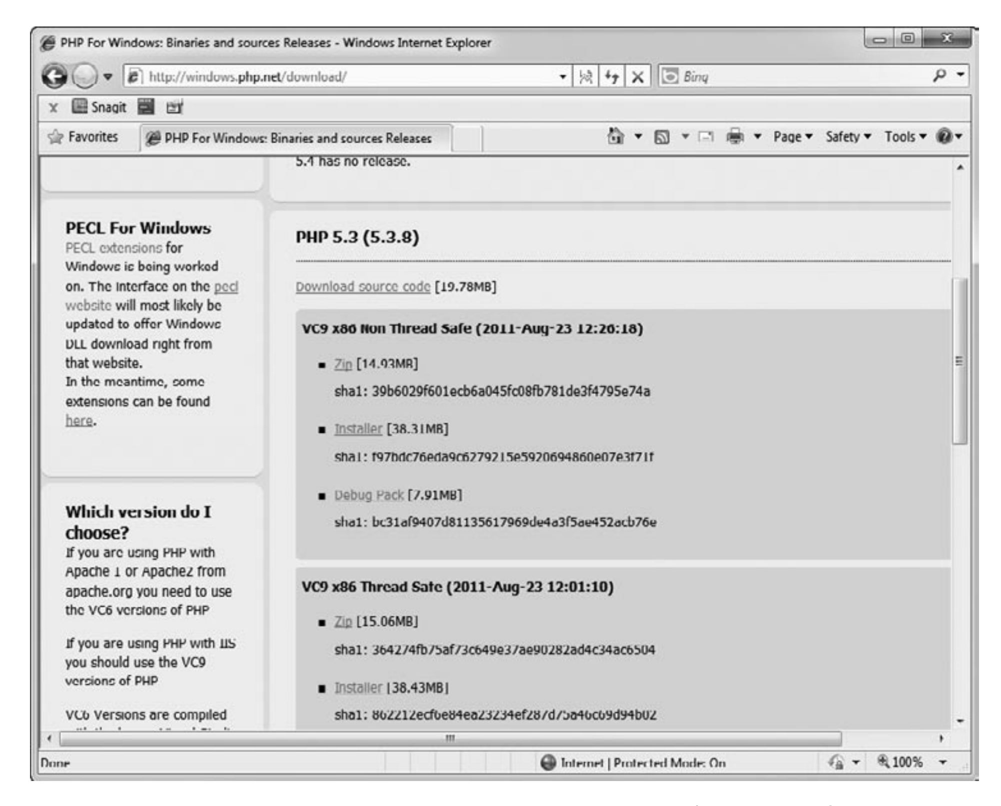


Рис. 1.3. Страница, посвященная загрузкам PHP для работы под Windows

К ВАШЕМУ СВЕДЕНИЮ

Релизы внутри версии

Если вам не приходилось работать с программным обеспечением, распространяемым в виде версий или релизов, то вам не о чем беспокоиться. Оба этих слова по отношению к программному обеспечению означают практически одно и то же: версия или релиз ПО — это всего лишь способ сообщить о том, что все его составляющие части собраны в один работоспособный пакет. Но поскольку программное обеспечение подвергается частым изменениям, команде разработчиков ПО нужен способ сказать: «В нашей программе появились новые свойства! Теперь вам доступен новый пакет!» Разработчики программы (а они действительно существуют) используют для этого номера версий (или номера релизов). Обычно сначала для программы указывается версия программы 1.0, и этот номер повышается с добавлением к программе новых свойств. Итак, версия 2.2 PHP будет новее версии 1.1 и, наверное, будет также иметь какие-нибудь новые полезные свойства. Иногда, как на веб-сайте PHP, вы увидите несколько различных пакетов или загружаемых частей программного обеспечения, у каждой из которых будет разный номер версии. Обычно вы можете просто загрузить самую последнюю версию требуемой вам программы — и будете абсолютно правы.

ПРИМЕЧАНИЕ

Если вы полагаете, что можете просто перейти непосредственно на страницу http://windows.php.net/download, то вы абсолютно правы: такой переход возможен. Но полгода спустя вы можете забыть столь длинный URL-адрес, но будете помнить www.php.net. Кроме того, старый добрый поисковик Google по ключевому слову PHP приведет вас на страницу www.php.net, поэтому совсем не лишним будет узнать, как добраться до Windows-установщика с главной страницы сайта, посвященного PHP.

Как только загрузка завершится, найдите загруженный файл и дважды щелкните на его значке. Когда Windows запросит разрешение на запуск установщика, щелкните на кнопке Allow (Разрешить), а затем в появившемся окне щелкните на кнопке Next (Далее), чтобы начать установку.

Вам нужно будет принять условия лицензионного соглашения, а затем выбрать каталог для установки. Согласитесь с предлагаемым каталогом C:\Program Files\PHP\, чтобы все ваши остальные программы всегда могли найти PHP. Затем установщик спросит насчет настроек веб-сервера (рис. 1.4). Пока вы будете использовать PHP на своей машине для тестирования программ, а затем будете выкладывать эти программы на веб-сервер, поэтому выберите пункт, позволяющий не настраивать веб-сервер (Do not setup a web server). Если позже вам понадобится добавить веб-сервер, вы всегда сможете вернуться назад и выбрать другой пункт.

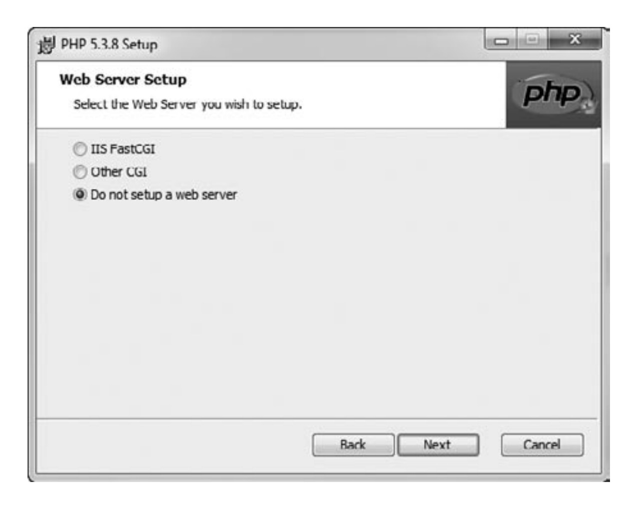


Рис. 1.4. Настройки веб-сервера

ПОД КАПОТОМ

PHP на Windows. Что выбрать: быстрый или безопасный?

PHP впервые был выпущен в версии, предназначенной для работы в Windows, в далеком 2000 году. В этих ранних релизах PHP был представлен только одной версией: безопасной для потоков (Thread Safe). В то время как Mac OS X и Unix/Linux-системы использовали некие механизмы, вызывающие процессы для запуска одновременно нескольких задач, система Windows применяла потоки. Эти Windows-потоки

могли взаимодействовать друг с другом, и, чтобы они не мешали друг другу, PHP поставлялся в версии, безопасной для потоков.

К сожалению, разведение потоков по индивидуальным путям занимает очень много времени. Безопасная для потоков версия PHP для Windows работает довольно медленно, и PHP-программисты стараются быть подальше от Windows везде, где это возможно. Несколько талантливых PHP-программистов нашли способы повторного использования потоков, и теперь множество веб-серверов, работающих под Windows, поставляются с предустановленной PHP-версией, способной повторно применять потоки с самого начала. Но не всем понравилось устанавливать PHP с необходимостью последующей инсталляции модифицированного веб-сервера или вручную вносить изменения в PHP, чтобы добиться от него приемлемой скорости работы. В результате теперь появилась возможность выбрать версию, небезопасную для потоков (Non Thread Safe). Данный вариант не отвечает за состояние других потоков, в результате чего он получил значительный прирост производительности: в диапазоне от 10 до 40 % в зависимости от характера ваших приложений.

Возможно, если у вас не будет твердой уверенности в том, какая версия двоичного кода PHP вам нужна, стоит выбрать двоичный код, небезопасный для потоков. В результате вы получите отличную производительность. Если же у вас есть серьезные опасения насчет версии, небезопасной для потоков (может быть, вам не хотелось бы, чтобы два пользователя состязались за получение одного и того же фрагмента данных), и вам не очень важно, насколько быстро будет работать ваше приложение, то в таком случае будет разумнее выбрать двоичный код, безопасный для потоков, и настроить собственную установку так, как вы считаете нужным.

Следующий экран задаст вопрос, что именно устанавливать (рис. 1.5). Инсталлятор Windows поставляется с базовой PHP-установкой, но вы можете добавить некоторые дополнительные компоненты, доступ к которым можно получить, щелкнув



Рис. 1.5. Окно выбора дополнительных компонентов

на белых прямоугольниках рядом с дополнениями (Extras) и выбрав отдельные функции. На данном этапе нас вполне устроит выбор, предлагаемый по умолчанию. Просто щелкните на кнопке Next (Далее), чтобы пройти дальше.

И наконец, щелкните на кнопке Install (Установить) и подождите, пока заполнится индикатор хода установки. Вот и все! Теперь на вашей машине есть работающий PHP.

Чтобы проверить PHP в работе, перейдите в меню Пуск и введите в поле поиска cmd. Откроется окно командной строки, в котором можно набирать команды, подобные тем, с помощью которых запускается PHP. Наберите в этом окне php (рис. 1.6).

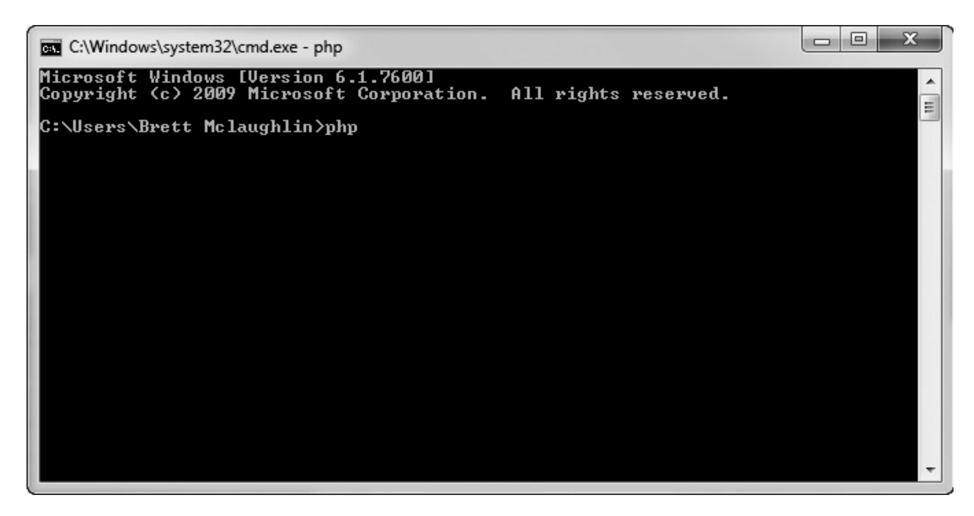


Рис. 1.6. Запуск PHP из командной строки

Если даже вы не увидите ничего особенного, эта пустая строка и приглашение на ввод команды означают, что PHP установлен правильно. Теперь вы готовы к созданию своей первой программы.

PHP на компьютерах Macintosh

Компьютеры Macintosh, столь внешне привлекательные, трудно представить себе инструментом программиста, работающего на PHP. Но, тем не менее, если у вас есть Macintosh, то вместе с ним вы получаете и предустановленный PHP. Чтобы убедиться в этом, откройте на Macintosh-компьютере приложение Terminal.

Если вы никогда не пользовались приложением Terminal, то ничего страшного, вы быстро привыкнете к нему и оно станет одним из ваших лучших друзей для работы с PHP. Откройте папку Applications (для этого можно воспользоваться сочетанием клавиш Shift+⌘+A), а затем найдите папку Utilities, показанную на рис. 1.7. В папке Utilities скрыты всевозможные полезные программы, поставляемые вместе

с Mac OS X. Присмотритесь к ним внимательнее, и вы явно найдете среди них полезные для себя программы, которые могут войти в ваш круг регулярного использования.



Рис. 1.7. Папка Utilities с полезными программами, поставляемыми вместе с Mac OS X

ВНИМАНИЕ

Сочетание клавиш Shift+⌘+A работает только в том случае, если активно окно Рабочего стола или другой файловой папки. Если сейчас вы просматриваете эту книгу в ридере, то щелкните для примера на Рабочем столе, а затем нажмите сочетание клавиш Shift+⌘+A.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Открытие программ без мыши

Вы, конечно, можете открыть папку в Finder с помощью мыши. Но можно все делать намного быстрее, не убирая руки с клавиатуры. Сочетания клавиш вроде Shift+⌘+A — прекрасный способ такой работы, поэтому стоит уделить время изучению таких команд. Обращайте на них внимание, когда они будут встречаться вам в меню некоторых программ или в главах этой книги.

В книге будут приводиться и пути к папкам, и сочетания клавиш, если таковые существуют. Пути к папкам будут показаны с помощью стрелок, например Applications ▸ Utilities (Приложения ▸ Утилиты).

Итак, откройте папку Applications (Приложения) и найдите приложение Terminal. Его значок похож на компьютерный терминал с черным экраном и небольшой белой стрелочкой (рис. 1.8). Программа Terminal позволяет использовать на компьютере Macintosh окно командной строки. Мы будем выполнять большой объем работы по программированию на PHP с помощью данной программы, поэтому вы быстро привыкнете к ней.

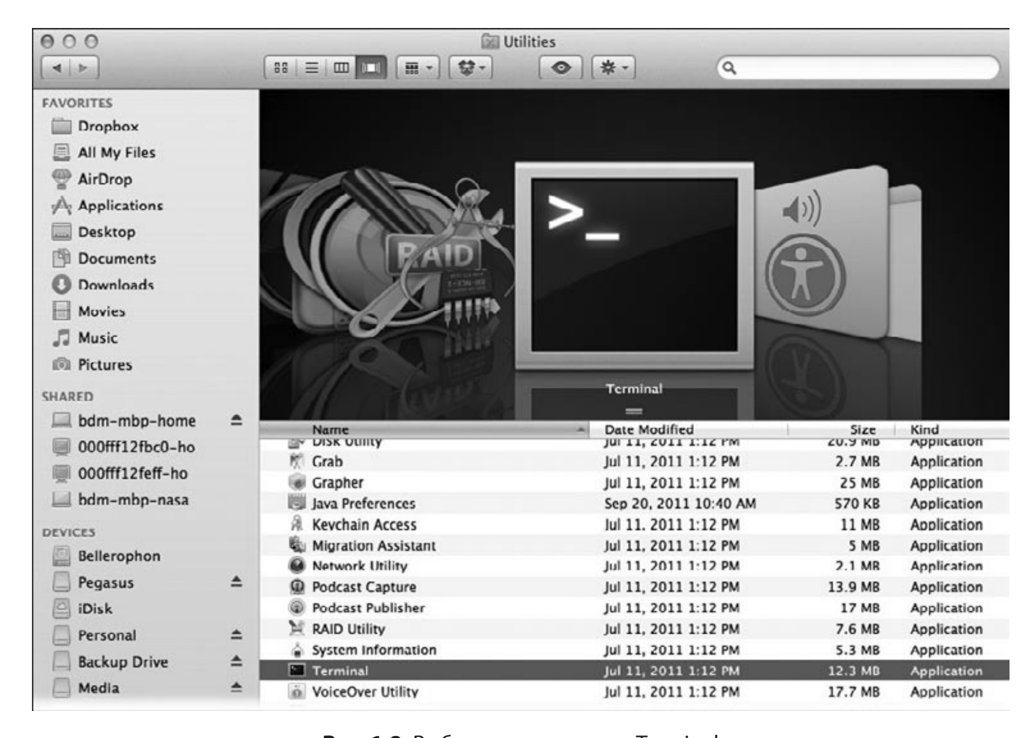


Рис. 1.8. Выбираем программу Terminal

СОВЕТ

Приложение Terminal будет часто использоваться для тестирования PHP-программ перед выкладыванием на сервер. Чтобы упростить запуск приложения Terminal, можно перетащить значок на dock-панель.

Дважды щелкните на значке программы Terminal. Появится белый прямоугольный экран с небольшим мигающим черным курсором (рис. 1.9). Да, когда окно программы Terminal будет открыто в первый раз, оно вряд ли сможет кого-то впечатлить. Вы получите строку, которая, возможно, будет соответствовать имени вашего компьютера, а затем таинственный знак доллара. Не волнуйтесь, вскоре вы со всем этим вполне освоитесь.

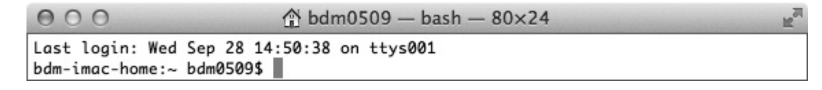


Рис. 1.9. Окно программы Terminal

Чтобы убедиться, что PHP установлен в вашей системе, просто наберите php, все слово строчными буквами, и нажмите Enter. В результате мигающий курсор должен превратиться в скучный темно-серый прямоугольник.

Чтобы убрать этот застывший прямоугольник, нажмите сочетание клавиш Control+C, и вы опять получите мигающий курсор. А теперь наберите which php. Команда which сообщит, где на вашем компьютере находится программа, имя которой было ей передано, то есть в данном случае — php. В ответ вы получите нечто похожее на изображение, приведенное на рис. 1.10. На нем показано, что файл php находится в каталоге /usr/bin. У вас, наверное, будет тот же результат.

```
bdm0509 — bash — 80×24
Last login: Wed Sep 28 14:50:50 on ttys002
bdm-imac-home:~ bdm0509$ php
^C
bdm-imac-home:~ bdm0509$ which php
/usr/bin/php
bdm-imac-home:~ bdm0509$
```

Рис. 1.10. Команда which позволяет узнать точное местонахождение программы на машине

Если вы увидите, где именно находится php, значит, все готово к работе!

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Контролирование установки PHP

Как и многие другие программы на вашем компьютере, программный пакет PHP (включающий программу php, которую вы уже запускали) довольно часто обновляется. В большинстве случаев, если вы загружаете обновления компьютера с помощью системы Apple Software Update, то вам не нужно специально беспокоиться об обновлении PHP. Но если нужно посмотреть, какую версию PHP вы только что запустили, можно в окне программы Terminal набрать команду php -version. При этом вы получите нечто подобное:

```
Bretts-MacBook-Pro:~ bdm0509$ php -version

PHP 5.3.4 (cli) (built: Dec 15 2010 12:15:07)

Copyright (c) 1997-2010 The PHP Group

Zend Engine v2.3.0, Copyright (c) 1998-2010 Zend Technologies
```

В первой строке, выданной PHP, сообщается, что запущена версия 5.3.4. (О номерах версий более подробно рассказано во врезке «Релизы внутри версии» раздела «PHP на персональном компьютере (PC)» данной главы.)

Если нужно получить самую свежую версию PHP, можно посетить веб-сайт www.php.net и загрузить исходный код PHP. Это немного сложнее использования версии, предустановленной на вашем Macintosh-компьютере, поэтому, если вы слабо разбираетесь в таких командах, как unzip и tar, можете по-прежнему пользоваться тем, что уже установлено на вашей машине.

Кстати, если вы пользуетесь Mac Software Update не слишком часто, то можете сделать это прямо сейчас. Это поддержит ваше программное обеспечение в актуальном состоянии без всей этой возни с самостоятельной загрузкой программ. Просто выберите команду ▸ Software Update (▸ Обновление программного обеспечения) для поиска и установки нового программного обеспечения, доступного для вашей Macintosh-машины. И если ваше программное обеспечение обновилось, то появится сообщение об этом.

Подбор текстового редактора

Все программы на PHP являются старыми добрыми тестовыми файлами. Фактически написание PHP-программы ничем не отличается от написания кода HTML, CSS или JavaScript. Вы набираете разный текст, но все это все равно текстовые файлы, сохраненные со специальными расширениями. Для HTML используется расширение HTML, для CSS — CSS, для JavaScript — JS, а для файлов PHP — расширение PHP.

Поскольку код PHP является простым текстом, для работы с ним понадобится хороший текстовый редактор. Каким бы простым ни был текстовый редактор, он вполне подойдет для программирования на PHP. Если вы работаете в Windows, то можете воспользоваться программой Блокнот. Если на Macintosh, то вам вполне подойдет программа TextEdit. Эти приложения уже установлены на вашем компьютере, и вам не нужно ничего загружать или покупать. Плохо только то, что ни одна из этих программ не знает, что вы пишете код PHP, поэтому она не поможет выявить опечатки или сформировать структуру ваших файлов.

Кроме этих программ существуют редакторы, предназначенные для работы с PHP. Например, в Windows вы можете загрузить редактор NuSphere PhpED (www.nusphere.com/products/phped.htm) (рис. 1.11).

За программы типа NuSphere придется немного раскошелиться, обычно их цена колеблется в диапазоне от $50 до $100, но при этом вы получите необычайную расцветку программного кода, помощь по специальным свойствам языка и во многих случаях весьма симпатичную структуру файла и даже возможность выкладывать PHP непосредственно на ваш веб-сервер, то есть решение задачи, рассмотренной далее в разделе «Выкладывание вашего кода HTML, CSS и PHP».

Если вы работаете на Macintosh, то вас могут заинтересовать две программы — BBEdit (www.barebones.com/products/bbedit/index.html) и TextMate (www.macromates.com). Оба эти редактора предназначены только для работы на Macintosh-машинах и предоставляют функции, сходные с теми, которые предлагает редактор PhpED, работающий в Windows: подсвечивание кода программы, управление файлами, справочную документацию, поддержку кода HTML, CSS, JavaScript, а также многое другое. Редактор BBEdit показан на рис. 1.12. За него придется выложить $100. BBEdit предоставляет намного больше полезных функций, чем просто адекватная

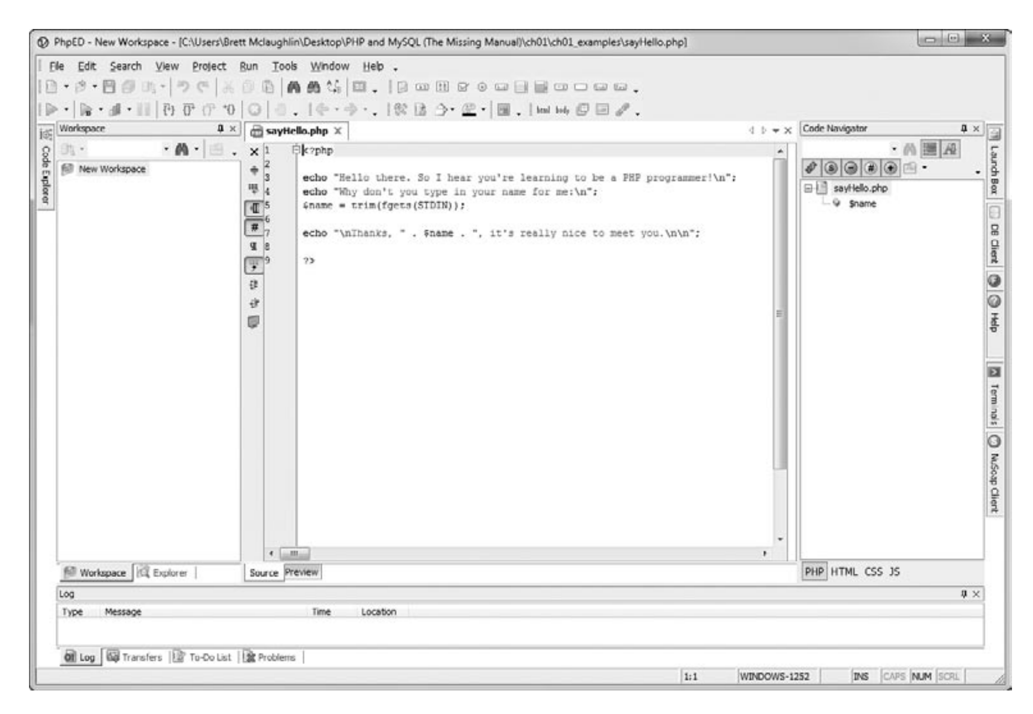


Рис. 1.11. Редактор NuSphere PhpED снабдит вас массой свойств и обеспечит поддержку кода JavaScript, CSS и HTML, а также поддержку кода PHP. Кроме этого он предоставит отличную документацию по большинству функций и библиотек PHP

поддержка PHP. Изначально он был настроен под HTML, поэтому в нем есть некоторые странности, но для работы с PHP на Mac-машине он является очень хорошим выбором.

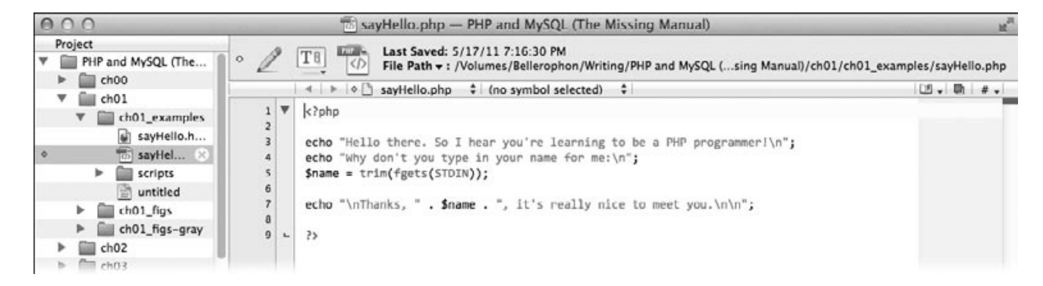


Рис. 1.12. Окно программы BBEdit

Окно редактора TextMate показано на рис. 1.13. Он выглядит проще, чем BBEdit. Если вы никогда раньше не пользовались редактором для программистов, то хорошо, наверное, будет начать именно с него. Он также предлагает возможность управления файлами и поддержку FTP, но лучше использовать его только для набора кода. Цена TextMate составляет примерно $60.

Рис. 1.13. Редактор TextMate предназначен для редактирования с цветовой подсветкой кода

ПОД КАПОТОМ

Текстовые редакторы: смешение программ

Хотя такие программы, как PhpED, BBEdit и TextMate, считаются текстовыми редакторами, на самом деле они представляют собой множество приложений, объединенных в одной оболочке. Представьте, что у вас есть текстовый редактор и диспетчер файлов, такой как Проводник Windows или Finder в Mac-системе, приложение telnet или программа Terminal, FTP-клиент и некое связующее их средство. Именно это вы получаете от таких программ: целый пакет собранных воедино средств.

Самое интересное в этих «текстовых редакторах плюс» — предлагаемые ими разнородные функции, благодаря которым вы не нуждаетесь в пяти или шести значках на dock-панели вашей Macintosh-машины или в таком же количестве ярлыков на Рабочем столе Windows. Вы получаете непосредственный доступ практически ко всему необходимому для создания веб-страниц или для программирования на PHP.

Однако не все так уж замечательно. Весь этот обобщенный инструментарий зачастую не дотягивает по качеству до полноценных отдельных инструментов. Иными словами, программа, пытающаяся делать абсолютно все, обычно справляется с множеством дел, но весьма посредственно. В то время как множество приложений выполняют всего одну задачу, но делают это по-настоящему хорошо.

Зачастую вам приходится выбирать между удобством и функциональностью. Если при необходимости выложить файлы на сервер вы пользуетесь только FTP, то вы практически никогда не станете работать со средствами командной строки, и если вам нравятся раскрашенные тексты исходного кода, то комбинированные текстовые редакторы с массой дополнительных функций могут стать для вас вполне достойным выбором.

Но независимо от того, какой именно текстовый редактор вы используете, с дополнительными функциями или без них, в какой-то определенный момент вам может потребоваться выйти из редактора и воспользоваться настоящими программами FTP или telnet. Если вам удобно время от времени переходить к работе с такими программами, не пользуясь для отправки файлов редактором, то без тени сомнения продолжайте вводить свой код в TextMate или в PhpED.

Как только вы освоитесь в написании кода PHP, не пожалейте времени и поработайте в разных текстовых редакторах, чтобы подобрать для себя лучший из них. Может также оказаться, что вам для программирования в конце концов оптимальнее Блокнот или TextEdit. Для PHP нет единого верного решения, может подойти любой из этих вариантов.

Сначала лучше попробовать поработать в простом текстовом редакторе: Блокноте в Windows или TextEdit на Macintosh. Вы узнаете многое о PHP, даже если не получите все эти функции редактора, предлагающего массу дополнительных свойств. Кроме того, как только вы по-настоящему поймете, что такое PHP, и научитесь использовать его свойства безо всякой автоматизации, вы оцените функции других редакторов и даже сможете намного эффективнее их применять.

ПРИМЕЧАНИЕ

Когда вы ознакомитесь с PHP, вы также сможете освоить Eclipse PHP (www.eclipse.org/projects/project.php?id=tools.pdt). Интегрированная среда разработки Eclipse IDE давно уже является излюбленным средством Java-разработчиков. А теперь в ней вполне достаточно дополнительных модулей и для PHP, чтобы сделать ее достойным выбором также и для PHP-программистов. Однако в Eclipse слишком много компонентов, масса инструментов и приспособлений, поэтому стоит немного подождать, прежде чем окунуться в нее с головой. Но чуть позже можно будет вернуться к этому средству и уделить ему внимание.

Создание вашей первой программы

У вас уже есть PHP, а также текстовый редактор. Теперь создадим нашу первую PHP-программу. Отройте текстовый редактор и наберите следующий код именно так, как он показан:

```
<?php

echo "Приветствую вас здесь. Слышал, вы учитесь на PHP-программиста!\n";
echo "Почему бы вам не набрать свое имя для меня:\n";
$name = trim(fgets(STDIN));

echo "\nСпасибо, " . $name . ", очень рад с вами познакомиться.\n\n";

?>
```

Многое в этом коде может показаться вам странным, но вы не должны этого пугаться. Совсем скоро перед вами откроются все его тонкости. А теперь просто посмотрим на PHP, который почти не отличается от HTML или от JavaScript.

ВНИМАНИЕ

Некоторые редакторы, которыми вы можете воспользоваться, например TextEdit, автоматически создают документы в сложном текстовом формате. Это формат позволяет использовать форматирование, например выделение полужирным шрифтом и подчеркивание. Но для нашего PHP-кода все это ни к чему. Поэтому поищите в нем настройку, включающую применение простого текста, который не допускает форматирования. При использовании TextEdit следует выбрать пункт меню Format ▸ Make Plain Text (Формат ▸ Простой текст). Этого пункта может и не быть, если вы уже набираете простой текст. При использовании Блокнота сложный текстовый формат недоступен, поэтому можно не волноваться.

Код PHP — это простой текст, но в нем используется несколько странных символов. Вам нужно привыкать к набору знака доллара ($), угловых скобок (< и >, так же, как в HTML) и обратного слэша (\). Вам придется пользоваться этими символами довольно часто (рис. 1.14).

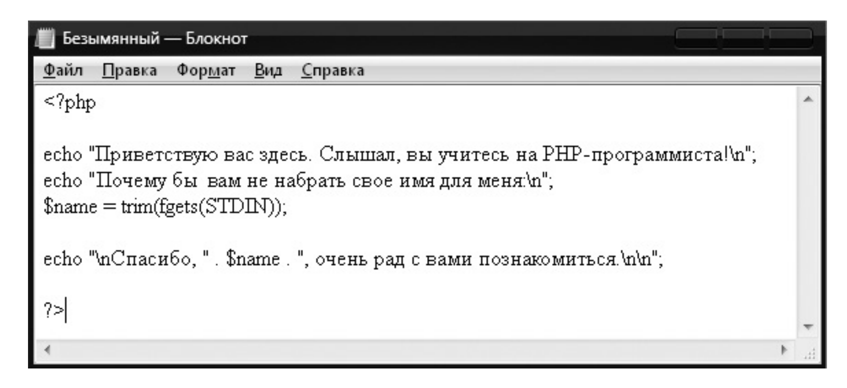


Рис. 1.14. Набранный код программы

Итак, наша программа выполняет несколько простых действий.

1. Идентифицирует саму себя как PHP-код с помощью фрагмента `<?php`.
2. Выводит приветственное сообщение, используя команду `echo`.
3. Просит посетителя ввести его имя, также применяя для этого команду `echo`.
4. Получает имя посетителя и помещает это имя в нечто по имени `$name`.
5. Обращается к посетителю, выводя сообщение, за которым следует то, что содержится в `$name`.
6. Завершает код символами `?>`.

Неважно, что многие фрагменты этого кода пока не имеют для вас никакого смысла. Но, возможно, во многом вы уже разобрались, за исключением, может быть, той странной строки, которая начинается с `$name =`. В коде также есть странные символы вроде `\ns` и `STDIN`, о которых вы вскоре узнаете. Просто попробуйте проследить порядок следования фрагментов кода: открывающий фрагмент `<?php`, вывод информации, запрос имени пользователя, еще один вывод информации и закрывающий фрагмент `?>`.

Теперь сохраним эту программу. Назовите ее `sayHello.php`. Не забудьте добавить ее расширение `.php`, иначе в конечном итоге возникнут проблемы. Сохраните файл в каком-нибудь удобном месте, например на **Рабочем столе**, в своем личном каталоге или в папке, которую вы будете использовать для хранения всех ваших PHP-программ в процессе учебы.

ПРИМЕЧАНИЕ

Большинство программ в Windows и Macintosh, в которых вы будете писать код, будут предоставлять вам по умолчанию при сохранении файла, например, расширение TXT. Обязательно поставьте вместо него расширение PHP. В Windows зачастую расширения файлов скрыты, поэтому убедитесь что полным именем файла является sayHello.php, а не что-нибудь вроде sayHello.php.txt.

Ну вот вы и написали свою первую PHP-программу!

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Использование простого текста по умолчанию

Большинство популярных текстовых редакторов позволяют переключаться между сложным текстовым форматом и простым текстом на основе структуры редактируемых файлов, но по умолчанию они начинают работу в режиме сложного текстового формата. Это может вызывать сильное раздражение, поэтому вы можете захотеть настроить свой редактор всегда начинать работу в режиме простого текста. Если вы работаете с редактором TextEdit на Mac-системе, откройте меню Preferences (Настройки). В окне Preferences имеется масса настроек, но сейчас важнее всего формат текста и шрифт, используемый для простого текста. В самой верхней части под заголовком Format (Формат) можно установить в качестве режима, используемого по умолчанию, Plain Text (Обычный текст) (рис. 1.15). При работе в Windows использование Блокнота позволяет избегать этих проблем, поэтому вам не о чем волноваться.

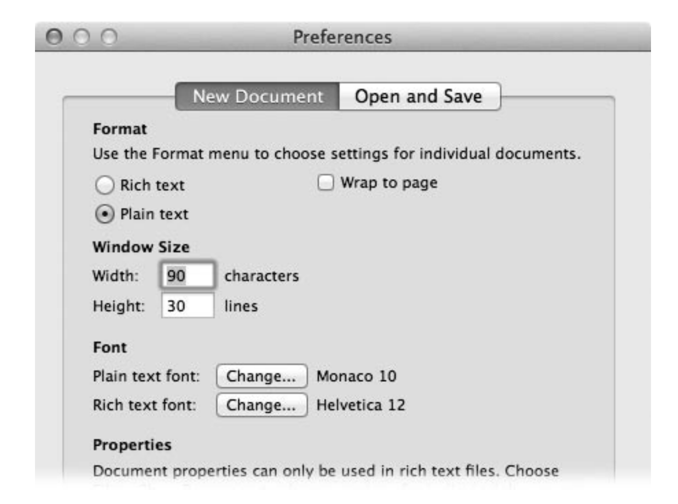


Рис. 1.15. В параметры TextEdit можно зайти через меню Preferences (Настройки) или с помощью сочетания клавиш ⌘+,

Запуск вашей первой программы

Какой прок от набранного кода, если его нельзя посмотреть в работе? Эта конкретная программа еще не готова к работе в сети, поэтому вам придется воспользоваться окном командной строки. Мы уже использовали это окно для проверки правильности установки PHP. Запустите программу командной строки еще раз. Если вы работаете на Mac-системе, то вы уже открывали Terminal и, возможно, у вас уже есть его значок на dock-панели. Откройте Terminal еще раз.

Теперь перейдите в каталог, в котором была сохранена ваша программа `sayHello.php`. Чтобы удостовериться, что вы в нужном каталоге, список каталогов можно вывести командой `dir` (в Windows) или `ls` (на Macintosh). Находясь в нужном каталоге, наберите в командной строке следующую команду:

```
php sayHello.php
```

Она прикажет запуститься программе `php` и передаст ей в качестве запускаемой вашу программу `sayHello.php`. Вы увидите набранное вами приветственное сообщение, а затем программа запросит ввод вашего имени. Наберите свое имя, а затем нажмите Enter. После этого программа поблагодарит вас, как показано на рис. 1.16.

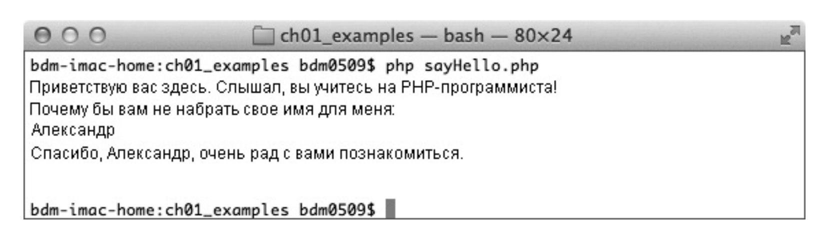


Рис. 1.16. Вывод информации в окне командной строки

Со временем вы будете запускать большинство своих сценариев PHP в веб-браузере. А пока управлять командой php, передавать ей конкретный сценарий для запуска, а затем наблюдать выводимую информацию позволит окно командной строки.

Вот и все! Ваша первая программа работает, и вы готовы окунуться в изучение PHP.

Создание вашей второй программы

Зачем ждать, прежде чем продолжить углубление в PHP? Ведь ваш интерес к PHP, скорее всего, обусловлен вашим желанием вывести свои веб-страницы за пределы возможностей, предоставляемых им JavaScript. Если я прав, то PHP — это замечательный язык, и вам нужно научиться выкладывать код, который вы уже написали, в Интернет. А поскольку доступ к большинству PHP-программ осуществляется через веб-страницы, довольно часто программирование на PHP начинается с создания HTML-страницы, которая будет отправлять информацию вашим PHP-сценариям.

Начало работы: создание HTML-страницы

Откройте в текстовом редакторе или в привычном для вас HTML-редакторе новый документ и создайте следующую HTML-страницу:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
```

```
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пример 1.1</div>

  <div id="content">
    <h1>Добро пожаловать!</h1>
    <p>Приветствую вас здесь. Слышал, вы учитесь на PHP-программиста!</p>
    <p>Почему бы вам не набрать свое имя для меня:</p>
    <form action="scripts/sayHelloWeb.php" method="POST">
      <p>
        <i>Введите свое имя:</i> <input type="text" name="name" size="20" />
      </p>
      <p><input type="submit" value="Поприветствуйте меня" /></p>
    </form>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

ПРИМЕЧАНИЕ

Как я уже говорил во введении, вы можете загрузить этот HTML, как и все остальные примеры, приведенные в данной книге, с веб-сайта www.missingmanuals.com/cds/phpmysqlmm. Вы можете также получить CSS и изображения, используемые в примерах, которые придадут вашим программам дополнительную привлекательность. Однако будет очень полезно (особенно, если вы только приступили к изучению PHP), если вы наберете код этих программ самостоятельно.

Для вас на этой странице не должно быть практически ничего нового. На ней всего лишь имеется ссылка на внешнюю таблицу стилей CSS, предоставляется текст приветствия, подобный тому, что использовался в нашей программе sayHello.php, а затем определяется форма, в которую пользователи могут вводить свои имена.

Единственное, что должно привлечь ваше внимание, — это следующая строка в определении формы:

```
<form action="scripts/sayHelloWeb.php" method="POST">
```

Этот код означает, что форма собирается отправить свою информацию программе под названием sayHelloWeb.php — PHP-программе, которую вы вскоре напишете. После отправки формы в действие вступит программа sayHelloWeb.php, которая выведет приветствие.

Создание PHP-сценария

После получения HTML-страницы, отправляющей информацию программе sayHelloWeb.php, нужно создать код этой программы. Написание PHP-кода для запуска в сети не имеет особых отличий от программы, которую вы уже создали в этой

главе. Информацию нужно получить немного другим способом, поскольку у вас нет командной строки, в которой пользователь может что-нибудь набрать. Но кроме этого весь остальной код практически такой же.

Откройте новое окно текстового редактора и введите следующий PHP-код (он должен быть чем-то похожим на превращенную в HTML версию программы sayHello.php):

```
<html>
 <head>
  <link href="../../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пример 1.1</div>

  <div id="content">
    <h1>Привет, <?php echo $_REQUEST['name']; ?>!</h1>
    <p>Рады приветствовать вас. Добро пожаловать в начало вашей одиссеи
    в мире PHP-программирования.</p>
    </form>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

Сохраните эту программу под именем sayHelloWeb.php и убедитесь, что файл имеет простой текстовый формат и правильное расширение.

Первое, что вам здесь, наверное, бросилось в глаза, это то, что файл в целом выглядит как код HTML. Фактически при сравнении с sayHello.php, первой написанной вами программой на PHP, эта версия по стилю программирования может показаться более легкой для изучения. Причина в том, что при использовании PHP для работы с веб-страницами и для взаимодействия с ними ваши PHP-программы будут в основном вставлять данные в существующие веб-страницы, а это значит, что вам предстоит большой объем работы с HTML. Это, разумеется, хорошая новость, поскольку вы уже знакомы с HTML. Поэтому все сведется к добавлению кода к тому, что вы уже знаете, а не к изучению чего-нибудь совершенного нового с самого начала.

Поняв, что основная часть этой программы является простым HTML-кодом, вы, наверное, уже догадались, чем эта основная часть занимается. Разберем ее подробнее.

- Страница начинается с обычного элемента html и раздела head.
- Раздел body начинается с задания заголовка страницы и номера примера, точно так же, как и на обычной HTML-странице sayHello.html.
- Заголовок страницы определяется элементом h1 и выводит строку «Привет,».

- Элемент <?php сообщает браузеру о наличии кода PHP. Затем осуществляется доступ к переменной $_REQUEST, и свойство по имени name внутри этой переменной выводится с использованием инструкции echo.
- Окончание PHP-кода обозначается элементом ?>.
- Весь остальной HTML-код выводится точно так же, как и в sayHello.html.

Эта программа, как и большинство PHP-приложений, которые вам еще предстоит создать, получает ввод с веб-страницы, либо из одного встроенного HTML-кода (как на ранее созданных вами страницах), либо из другой PHP-программы. Задача, возлагаемая на такую веб-страницу (в нашем случае — на sayHello.html), заключается в том, чтобы побудить пользователя ввести сведения о себе, а затем отправить полученную информацию этой программе. Информация с этой HTML-страницы сохраняется в $_REQUEST, которая является в PHP специальной переменной.

Изменчивость переменных

Переменная в PHP, как в любом другом языке программирования, является хранилищем значения. У переменных есть имена. В PHP эти имена могут быть практически любыми. Узнать переменную в PHP можно благодаря тому, что ее имя начинается с символа $. Например, $myHeight — это переменная по имени myHeight, а $_REQUEST — это переменная по имени _REQUEST.

ПРИМЕЧАНИЕ

С технической точки зрения имя PHP-переменной не включает в себя символ $, но большинство PHP-программистов считают, что $ является частью самой переменной. Поэтому вы можете услышать, что для ссылки на $myHeight PHP-программисты говорят «доллар май хайт», а не «май хайт».

Но переменные — это не только имена. У них еще есть значение. Значением $myHeight (мой рост) может быть число 68 (означающее 68 дюймов[1]) или текст «68 дюймов». Но в PHP это значение не дается раз и навсегда. Значение переменной можно изменить, именно поэтому она называется *переменной*: переменная варьируется, или изменяется.

В программе sayHelloWeb.php для получения имени пользователя, введенного им в форму, которая была создана в файле sayHello.html, применяется специальная PHP-переменная $_REQUEST. PHP позволяет вам получить все, что ввел пользователь в форму, используя $_REQUEST и имя поля ввода формы, в данном случае — name (имя). То есть $_REQUEST['name'] возвращает информацию, помещенную пользователем в веб-форму, а именно в поле ввода по имени name (имя). Если пользователь также указал номер своего телефона, скажем, в поле формы по имени phoneNumber (номер телефона), в PHP это значение можно получить с помощью переменной $_REQUEST['phoneNumber'].

[1] 68 дюймов примерно равны 173 см. — *Примеч. ред.*

ПРИМЕЧАНИЕ

Если вы еще не вполне разобрались в том, как работают переменные и переменная $_REQUEST, ничего страшного. В нескольких следующих главах вы получите дополнительные сведения о переменных и, в частности, о специальных переменных в PHP, таких как $_REQUEST.

Как только PHP-программа получает значение из поля формы name (имя), она выводит это значение, используя инструкцию echo, которая уже применялась в вашей первой PHP-программе. Это значение попадает непосредственно в HTML-код, отправляемый обратно браузеру.

Выкладывание кода HTML, CSS и PHP

Если вы запускаете PHP-программу на своей машине с использованием окна командной строки, то как только код PHP будет сохранен, его можно будет запустить. Но при работе с веб-страницами и веб-приложениями все получается немного сложнее.

При создании веб-страницы вам нужно выложить любой написанный вами код (HTML, CSS, JavaScript) на собственный веб-сервер. Затем вы обращаетесь к таким файлам с помощью браузера посредством веб-адреса вида www.yellowtagmedia.com/sayHello.html. Ввод этого веб-адреса в браузер заставляет сервер предоставить HTML тому веб-браузеру, который запросил страницу.

PHP работает точно так же. Как только ваша PHP-программа будет написана, вы выкладываете ее на свой веб-сервер вместе со своим кодом HTML и CSS. При этом, как правило, получаются следующие файлы и каталоги.

- **Корневой, или исходный, каталог (/).** Это веб-корень, куда помещается весь ваш HTML-код. Обычно это то самое место, на которое ссылается такой URL-адрес, как yellowtagmedia.com/, без указания какого-либо конкретного файла после имени веб-сервера.
- **Каталог CSS (css/).** Это каталог, где хранятся все таблицы CSS.
- **Каталог JavaScript (js/).** Сюда помещаются файлы JavaScript. Часто этот каталог также называется scripts/, но, поскольку PHP-программы также вызывают сценарии (scripts), лучше давать каталогам более явные имена.
- **Каталог PHP (scripts/).** Сюда помещаются все PHP-программы. Этот каталог можно также назвать более конкретно, например php/ или phpScripts/, но в большинстве случаев веб-сайты используют для этого каталога название scripts/, поэтому применение именно этого названия считается правилом хорошего тона.
- **Каталоги примеров (ch01/, ch02/ и т. д.).** Поскольку вы работаете с примерами, у вас быстро накопится большое количество PHP-программ. Чтобы все было систематизировано, нужно для каждой главы иметь свой собственный каталог. Поэтому sayHello.html и sayHelloWeb.php следует выложить по адресу ch01/sayHello.html и ch01/scripts/sayHelloWeb.php.

ПРИМЕЧАНИЕ

Подобная организация не является обязательной, но если следовать ей, то все примеры, загружаемые для этой книги, будут работать без всяких изменений. Если вы измените структуру каталогов, то придется изменить все ссылки на CSS, JavaScript и другие PHP-программы в коде HTML и PHP.

Теперь, когда у вас уже есть готовый HTML- и PHP-код, вам нужно выложить эти файлы в правильные каталоги. Следует также загрузить файл `phpMM.css` с веб-сайта книги www.missingmanuals.com/cds/phpmysqlmm и поместить таблицу CSS в нужное место.

Когда все будет на своих местах, структура каталогов вашего веб-сервера приобретет вид, показанный на рис. 1.17. HTML- и PHP-файлы созданы нами конкретно для данной главы, именно поэтому они находятся в каталоге ch01/. А файл `phpMM.css` предназначен для примеров, создаваемых во всей книге, поэтому его нужно поместить в каталог css/, который следует сразу же за корневым каталогом веб-сервера.

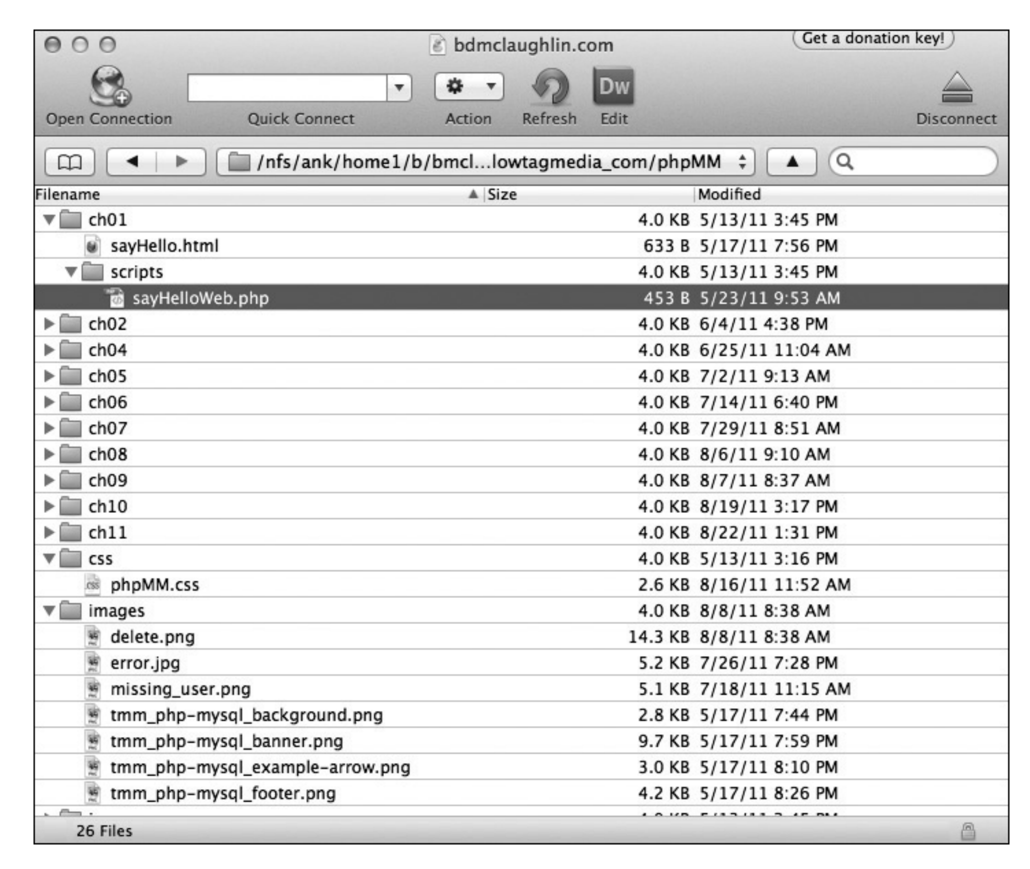


Рис. 1.17. Структура каталогов веб-сервера

Запуск вашей второй программы

Теперь HTML- и CSS-код находятся в нужных местах, а в HTML-форме в качестве действия указана ваша PHP-программа. В каталоге ch01/scripts/ также должен быть файл sayHelloWeb.php. Осталось только привести PHP в действие. Откройте веб-браузер и перейдите на ваш веб-сервер, а затем добавьте к имени сервера ch01/sayHello.html. Вы увидите созданный вами в файле sayHello.html HTML, похожий на изображение, которое представлено на рис. 1.18.

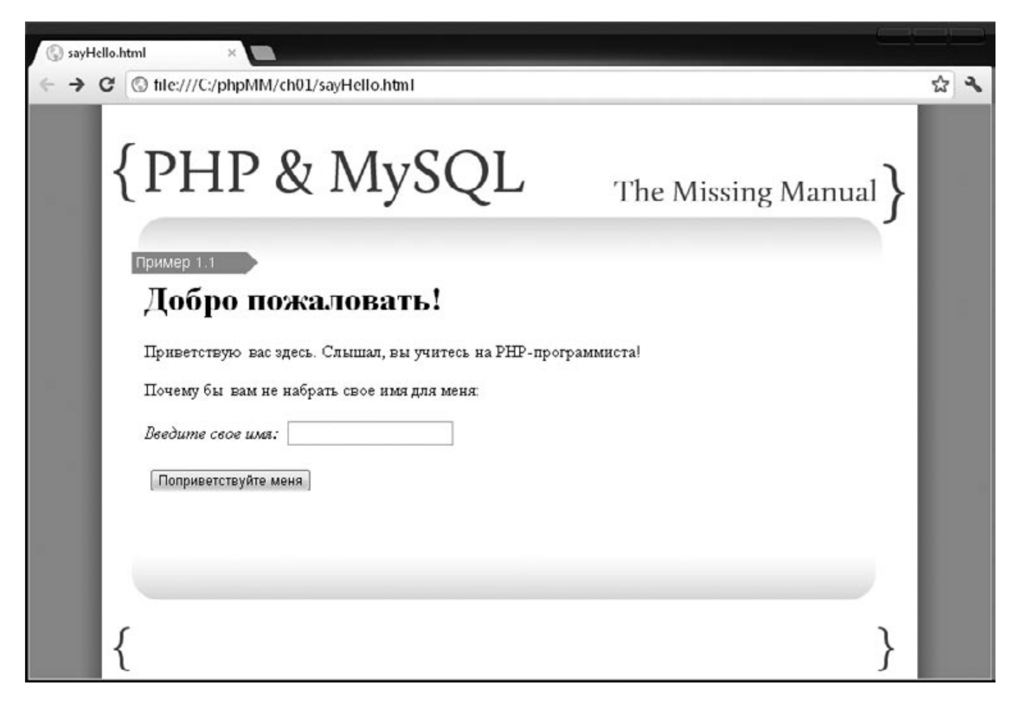


Рис. 1.18. Чаще всего вы обращаетесь к HTML-странице, а не непосредственно к PHP-программе. Но на этих HTML-страницах будут использоваться ваши PHP-программы для генерирования ответов на запросы ваших пользователей

Введите свое имя, а затем щелкните на кнопке Поприветствуйте меня. Страница отправит ваше имя как часть данных формы действию формы вашей программы sayHelloWeb.php. Эта программа запускается на вашем веб-сервере, а затем вам возвращается ответ (рис. 1.19). Таким образом, веб-браузер на самом деле не выполняет вашу программу. Вместо этого он заставляет запускать программу ваш сервер, и затем этот сервер возвращает результат запуска программы sayHelloWeb.php назад браузеру, который показывает вам персонифицированное приветствие.

Может показаться, что на вывод браузером вашего имени ушло слишком много работы. Фактически при желании можно было бы, наверное, написать подобную программу на JavaScript. Но создав несколько PHP-программ, вы должны понять, насколько просто пишется подобный код.

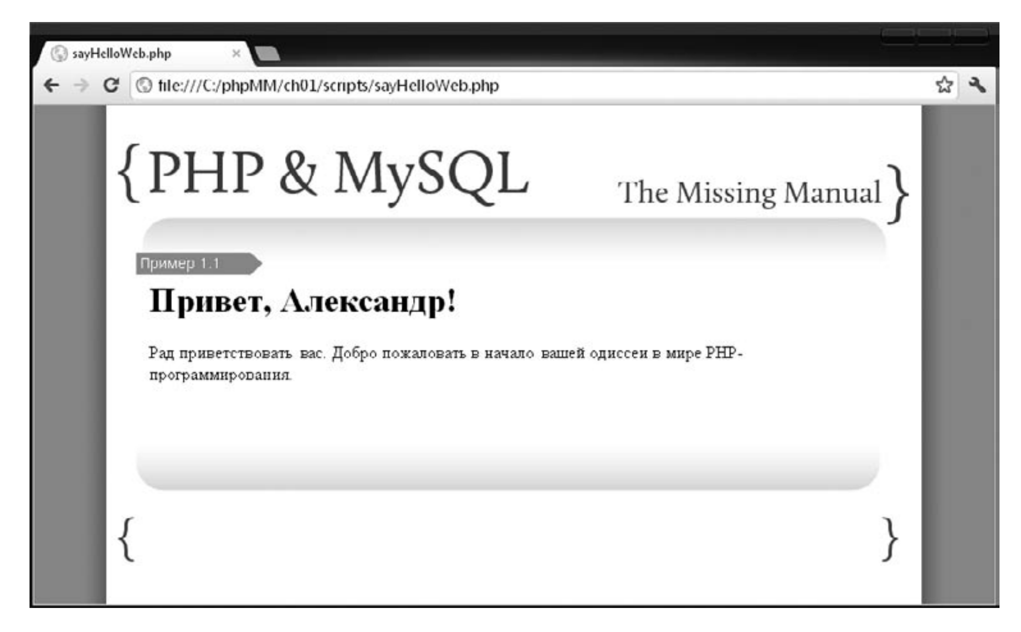


Рис. 1.19. Результат выполнения программы

И перед тем, как вы это узнаете, будет сделана намного более важная работа, чем сообщение пользователям их имен. Вы будете обращаться к базе данных, совершать сложные вычисления, принимать решения на основе сведений, предоставляемых вам пользователем, и той информации, которая хранится в базе данных, а также выполнять многое другое. Но все это начинается с небольшого HTML-фрагмента, PHP-программы, подобной той, что вы только что написали, и имеющейся структуры каталогов.

2 Синтаксис PHP: удивительный и таинственный

У нас уже есть две функционирующие PHP-программы, и одна из них даже работает с HTML-формой. Но пока вы просто набирали код. Даже если вы только приступили к изучению PHP, вы уже готовы копнуть поглубже и начать разбираться во всем происходящем в этом коде. В данной главе мы собираемся поближе познакомиться с синтаксисом PHP. Это предполагает изучение тех специальных слов, которые вы набирали в своих программах (их еще называют *ключевыми*).

К счастью, это изучение не означает, что вы не сможете по-прежнему создавать интересные программы, запускаемые в веб-браузере. Фактически, поскольку почти все, что делается с помощью PHP, касается веб-страниц, все ваши сценарии в данной главе будут принимать информацию из веб-формы и работать с этой информацией. Поэтому вы будете не только изучать PHP, но и учиться создавать веб-приложения.

Получение информации из веб-формы

В программе `sayHelloWeb.php` для получения значения переменной `name` из веб-формы `sayHello.html` использовалась следующая строка:

```
echo $_REQUEST['name'];
```

Возможно, вы запомнили, что `$_REQUEST` — это специальная PHP-переменная, позволяющая получать информацию из веб-запроса. В данном случае она используется для получения одной конкретной части информации — имени пользователя, но она может принести значительно больше пользы.

Непосредственный доступ к параметрам запроса

Чтобы узнать, насколько действительно полезна переменная `$_REQUEST`, отройте текстовый редактор, наберите показанный далее код, в котором посетитель вводит свое имя и несколько других важных сведений о себе (идентификатор в Twitter, URL-адрес страницы в Facebook и адрес электронной почты):

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пример 2.1</div>

  <div id="content">
    <h1>Вступайте в наш виртуальный клуб</h1>
    <p>Пожалуйста, введите ниже свои данные для связи в Интернете:</p>
    <form action="scripts/getFormInfo.php" method="POST">
      <fieldset>
        <label for="first_name">Имя:</label>
        <input type="text" name="first_name" size="20" /><br />
        <label for="last_name">Фамилия:</label>
        <input type="text" name="last_name" size="20" /><br />
        <label for="email">Адрес электронной почты:</label>
        <input type="text" name="email" size="50" /><br />
        <label for="facebook_url">URL-адрес в Facebook:</label>
        <input type="text" name="facebook_url" size="50" /><br />
        <label for="twitter_handle">Идентификатор в Twitter:</label>
        <input type="text" name="twitter_handle" size="20" /><br />
      </fieldset>
      <br />
      <fieldset class="center">
        <input type="submit" value="Вступить в клуб" />
        <input type="reset" value="Очистить и начать все сначала" />
      </fieldset>
    </form>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

СОВЕТ

Дополнительная информация о том, как в этом коде используется HTML, приведена в следующей врезке.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

HTML должен быть семантически значимым

Вы могли заметить некоторые весьма существенные изменения в этом HTML по сравнению с простой формой из главы 1. Там для разделения надписей и полей ввода в форме используются теги <p> и названия формы вручную отформатированы с помощью тегов <i>. Это соответствовало поставленной задаче, но не служило хорошим примером использования HTML.

При написании HTML-кода вы обычно структурируете свою страницу. Поэтому тег `form` не делает ничего, что имело бы визуальное отображение, он просто дает браузеру понять: «Здесь находится форма». Но при использовании таких тегов, как `<i>`, вы не описываете структуру, а сообщаете браузеру, как именно что-то должно выглядеть. Однако это не является настоящей задачей HTML, этим занимается CSS.

А в данной форме все форматирование было убрано за ее пределы. Вместо этого все надписи идентифицируются с помощью элемента `label` и атрибута `for`. Тем самым надписи идентифицируются именно как надписи, независимо от того, как они выглядят в конечном итоге, и это также связывает каждую надпись с конкретным полем ввода, которому она соответствует. В форме также имеется элемент `fieldset`, охватывающий разные блоки внутри формы: один для надписей и текстовых полей, а второй для кнопок формы. Эта структура также предоставляет семантическую информацию, имеющую вполне определенное значение. Путем придания HTML конкретного значения, браузеру (и другим реализаторам HTML) предоставляется информация, что есть что в нашей форме: если есть надписи (`labels`), стало быть, они нужны, чтобы обозначить название чего-то. Поля сгруппированы вместе с помощью тегов набора полей — `fieldset`. А оформление курсивом и полужирным шрифтом возложено на таблицы CSS, как это и должно быть. По-настоящему замечательно то, что в данном случае таблицы CSS смогут справиться со стилевой обработкой вашей формы значительно лучше. Поскольку вы избавились от форматирования в HTML, можно придать всем надписям формы одинаковый стиль, например выделить их полужирным шрифтом, задать выравнивание по правому краю и добавить отступ справа в 5 пикселов. То же самое применимо и к наборам полей. Вы можете задать рамку вокруг взаимосвязанных полей, что, собственно, и происходит при применении CSS к этой форме. Чтобы увидеть, как CSS влияет на эти HTML-элементы, посмотрите на рис. 2.1.

Если вы еще никогда не занимались приданием своим страницам семантической значимости, то привычка использовать HTML только для придания структуры и хранения всех стилевых установок в CSS выработается довольно быстро. Продолжайте работать в этом направлении, и ваши страницы станут выглядеть лучше. И все, кому придется обновлять ваши страницы в последующем, скажут вам спасибо.

Сохраните созданный файл под именем `socialEntryForm.html`. Чтобы убедиться в том, что HTML ведет себя подобающе, выложите его на сервер в каталог `ch02/`. Удостоверьтесь в том, что таблица CSS, предлагаемая с материалами по данной книге, находится в нужном месте — в каталоге `css/`, который, в свою очередь, располагается в корневом каталоге вашего сервера, — а затем откройте браузер и перейдите к HTML-форме. Изображение должно быть похожим на то, что показано на рис. 2.1.

Для получения информации, отправленной формой в `sayHelloWeb.php`, использовалась переменная `$_REQUEST`, а затем запрашивалось конкретное значение для имени — `name`. Но это новая форма, и из нее передается намного больше информации.

Рис. 2.1. Типовая страница ввода, заполняемая пользователем

Для получения всей этой информации нужно создать новый сценарий под названием getFormInfo.php и ввести в него следующий код:

```
<html>
 <head>
  <link href="../../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>
 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пример 2.1</div>

  <div id="content">
    <p>Это запись той информации, которую вы отправили:</p>
    <p>
      Имя: <?php echo $_REQUEST['first_name']; ?><br />
      Фамилия: <?php echo $_REQUEST['last_name']; ?><br />
      Адрес электронной почты: <?php echo $_REQUEST['email']; ?><br />
      URL-адрес Facebook: <?php echo $_REQUEST['facebook_url']; ?><br />
      Идентификатор в Twitter: <?php echo $_REQUEST['twitter_handle']; ?><br />
    </p>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

ПРИМЕЧАНИЕ

Если нужно усилить контроль над вашими сценариями, эту программу можно назвать не getFormInfo.php, а как-нибудь по-другому. Кроме того, нужно обеспечить обновление socialEntryForm.html и изменить значение атрибута формы action, чтобы оно соответствовало имени вашего сценария.

Рассмотрим, что происходит в этом коде. Кроме извлечения значений полей `first_name` и `last_name`, которое похоже на получение значения поля `name` в `sayHelloWeb.php`, используется переменная `$_REQUEST` для помещения значений, которые введены пользователем, в поля другой формы.

Вернитесь в свою веб-форму и введите вашу информацию. Затем отправьте форму — и увидите результат запуска `getFormInfo.php` (рис. 2.2).

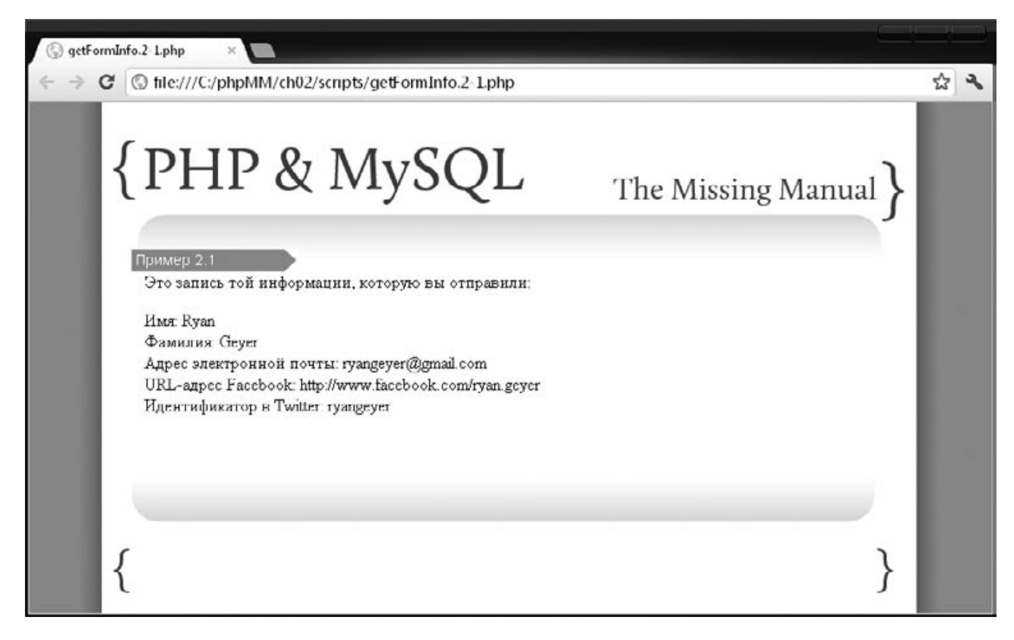


Рис. 2.2. Почти все в PHP начинается с информации, отправленной либо через веб-форму HTML, либо через другой PHP-сценарий

Фактически переменная `$_REQUEST` будет таким же образом использоваться в большинстве ваших PHP-программ:

```
echo $_REQUEST['FORM_INPUT_FIELD_NAME'];
```

Создание собственных переменных

Иногда нужно не только отправить значение поля. Вспомним нашу первую программу `sayHello.php` (ту версию, которая не запускалась в сети).

В этой программе создавалась собственная переменная:

```
$name = trim(fgets(STDIN));
```

PHP позволяет создавать какие угодно переменные. Нужно лишь подобрать описательное имя (подробнее об этом в следующей врезке «Для тех, кто понимает с полуслова. А что должно быть в имени? Много чего!») и поставить перед именем знак доллара:

```
$numberSix = 6;
$thisIsMyName = "Brett";
$carMake = "Honda";
```

Запомнив, как выглядит этот код, вернитесь к вашей новой программе getFormInfo.php. Вместо использования инструкции echo для вывода отправленной информации сохраните каждый ее фрагмент в переменной. Затем можно будет распоряжаться этой информацией как угодно и сколько угодно раз.

```
<?php

$first_name = $_REQUEST['first_name'];
$last_name = $_REQUEST['last_name'];
$email = $_REQUEST['email'];
$facebook_url = $_REQUEST['facebook_url'];
$twitter_handle = $_REQUEST['twitter_handle'];

?>

<html>
 <head>
  <link href="../../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
    <!-- Существующий HTML-код -->
  </body>
</html>
```

Обратите внимание, что блоки PHP-кода, начинающиеся с <?php и заканчивающиеся ?>, можно вставлять куда угодно. В этом сценарии на данный момент имеется блок PHP перед каждым HTML и еще несколько небольших блоков PHP внутри крупных фрагментов HTML. Когда и куда ставить PHP-код, решать вам, лишь бы он выполнял поставленную задачу.

Этот блок PHP-кода можно было бы поместить между открывающим страницу элементом html и элементом head или между элементами head и body. Выбор за вами.

ПРИМЕЧАНИЕ

То, что вы можете что-то сделать, еще не означает, что вы должны это делать. Зачастую лучше всего перед выводом какого-нибудь кода HTML проделать как можно больше работы с помощью кода PHP, а затем уже в одном месте вывести как можно больше HTML. Такая система позволит сосредоточить основную часть кода в одном месте, а основную часть HTML-вывода — в другом месте. (Дополнительные аргументы о важности поддержания хорошей организации кода изложены в следующей далее врезке «Для тех, кто понимает с полуслова. Рефакторинг кода в процессе его создания».)

Разумеется, остается еще множество мест, где PHP вставлен в HTML, как, например, в getFormInfo.php, и это вполне допустимо. Эти небольшие фрагменты PHP помещаются в HTML, и, конечно же, их не следует путать с 20 или 30 строками PHP-кода, вставленными в середину HTML.

Можно проверить внешний вид формы в браузере, но ничего нового, по сравнению с рис. 2.2, вы там не увидите. Дело в том, что ваш HTML-код, то есть та часть сценария, которую браузер показывает пользователю, не изменилась.

Но теперь в программе появился избыточный код. Сначала значение каждого поля формы получено из переменной $_REQUEST в блоке PHP-кода, который находится перед всем вашим кодом HTML, а затем те же самые значения этой переменной снова извлекаются уже в самом коде HTML. Когда одно и то же делается дважды, напрасно тратятся ценные ресурсы веб-сервера.

Но эта проблема легко устранима. Все нужные значения уже хранятся в переменных $first_name, $last_name и т. д. Поэтому в HTML-части файла getFormInfo.php можно просто с помощью echo вывести значения этих переменных, не используя $_REQUEST еще раз. Для этого нужно обновить div-контейнер с идентификатором "content":

```
<div id="content">
  <p>Это запись той информации, которую вы отправили:</p>
  <p>
    Имя: <?php echo $first_name; ?><br />
    Фамилия: <?php echo $last_name; ?><br />
    Адрес электронной почты: <?php echo $email; ?><br />
    URL-адрес в Facebook: <?php echo $facebook_url; ?><br />
    Идентификатор в Twitter: <?php echo $twitter_handle; ?><br />
  </p>
</div>
```

ДЛЯ ТЕХ, КТО ПОНИМАЕТ С ПОЛУСЛОВА

А что должно быть в имени? Много чего!

Вообще-то PHP не требует применения описательных имен. В Интернете можно встретить тысячи PHP-программ, код которых имеет следующий вид:

```
$x = $_REQUEST['username'];
$y = $_REQUEST['password'];
```

Этот код выполняется точно так же, как и код, использующий более описательные имена:

```
$username = $_REQUEST['username'];
$password = $_REQUEST['password'];
```

Причем многие программисты резонно заметят, что набор длинных описательных имен только прибавит работы. И это правда.

Но, с другой стороны, представляете, сколько сил будет потрачено на поиски всех мест появления переменной, хранящей имя пользователя во фрагменте кода, написанном вами или кем-нибудь еще три месяца назад? Предположим, что где-нибудь в отдаленном фрагменте сценария будет строка следующего вида:

```
echo "С возвращением на сайт, " . $y;
```

Неожиданно выясняется, что не так-то легко разобраться, что хранилось в переменной $x, а что в переменной $y. Где было имя пользователя: в $x или в $y?

Лучше перестраховаться, никому ведь не хочется, чтобы его пароль был выведен вместо имени пользователя!

Применение описательных имен, несмотря на то, что они длиннее и требуют больше времени для набора, упрощает чтение кода и вам, и кому-нибудь еще, кому может понадобиться в нем разобраться в процессе использования.

Вы можете еще раз отправить значения в свой файл `socialEntryForm.html` и убедиться в работоспособности обновленного сценария. Результат должен быть точно таким же, что и раньше (сравните его снова с тем, что показано на рис. 2.2). И пусть вас не удивляет, что вся эта работа проделана для получения точно такого же результата, поскольку фактически был выполнен большой шаг к грамотному программированию. Более подробно этот подход к программированию рассмотрен в следующей врезке. Усовершенствование в данной версии заключается в том, что все отправленные значения находятся в переменных.

ДЛЯ ТЕХ, КТО ПОНИМАЕТ С ПОЛУСЛОВА

Рефакторинг кода в процессе его создания

При каждом изменении кода, особенно с целью его упорядочения или выделения различных фрагментов по характеру их поведения, вы занимаетесь его рефакторингом, то есть улучшением его внутренней структуры с сохранением функциональности. Поэтому когда вы создаете PHP-блок в начале файла `getFormInfo.php` для сбора всей информации из отправленной формы, а затем просто выводите с помощью инструкции `echo` значение каждой переменной внутри HTML, вы занимаетесь рефакторингом своего сценария.

Рефакторинг может касаться как части какой-нибудь подпрограммы, так и целых файлов. Нужно пользоваться любой возможностью улучшить организацию сценария или, как будет сделано чуть позже, организацию нескольких совместно работающих сценариев. Даже если нет уверенности в том, что улучшение организации сможет помочь программе, им все равно стоит заняться. Когда спустя неделю, месяц или даже год вы вернетесь к своему коду, вспомнить о предназначении всех его составляющих будет значительно труднее. Еще хуже, если будет трудно вспомнить, что где в вашем сценарии находится. (К тому же вскоре ваши сценарии станут значительно длиннее.)

Проводя рефакторинг в процессе разработки, вы обеспечиваете простоту понимания того, чем занимается сценарий, при беглом взгляде на код. Это также означает, что при необходимости внесения изменений можно будет перейти непосредственно к тому месту, куда их нужно внести, проделать необходимую для этого работу и вернуться к размеренной жизни PHP-программиста.

Но имейте в виду: рефакторинг — обычно не самый веселый способ скоротать пятничный вечер. В большинстве случаев его целью не является изменение характера работы кода и особенно информации, выводимой в браузер. Поскольку речь идет о переделке, а иногда и об оптимизации (организации наиболее гладкого

☞

течения событий), целью является сохранение неизменными всех внешних проявлений.

Именно так все и происходит в случае с рефакторингом файла `getFormInfo.php`. Вы добавляете некий код PHP, создаете группу переменных, а затем применяете эти переменные в своем коде HTML. А что происходит с результатом? Он остается точно таким же, как и при использовании той версии, которая не подвергалась рефакторингу. Но теперь ваш код намного проще понять, и вы собираетесь получить некие несомненные преимущества от доступности данных переменных, что и будет показано далее.

Но зачем помещать значения в переменные? Именно сейчас это выглядит немного глупо: все, что делается, касается изменения того места внутри сценария, где извлекается информация из переменной `$_REQUEST`. Это не приносит никакой практической пользы. Что же можно сделать с этими переменными после того, как в них помещена информация? PHP предоставляет вам множество вариантов, особенно в том случае, когда речь идет о переменных, в которых содержится текст.

Работа с текстом в PHP

В PHP текст рассматривается всегда одинаково: просто как коллекция символов. Эти символы могут быть буквами, цифрами, пробелами, знаками пунктуации или чем-нибудь еще. И в PHP английское слово caterpillar является простой частью текста, как и что-либо абсолютно бессмысленное вроде !(gUHa8@m.@. Для вас же первое похоже на слово, а второе — на ругань отрицательного персонажа из игры QBert[1]. А для PHP все это просто текст. Фактически в PHP и в большинстве других языков программирования имеется специальное слово для ссылки на текст, поскольку он является важной частью языка: *строка (string)*. Стало быть, фрагмент текста может быть также назван строкой, и вместо слов «текстовый поиск» и «соответствие тексту» можно часто услышать, как программисты говорят о поиске строки или о соответствии строке.

Объединение текста

Положительная сторона того, что для PHP весь текст одинаков, — с ним можно производить массу интересных манипуляций, независимо от того, чем этот текст является. Например, в нашем сценарии `getFormInfo.php` есть пять переменных, каждая из которых заполнена текстом:

```
$first_name = $_REQUEST['first_name'];
$last_name = $_REQUEST['last_name'];
$email = $_REQUEST['email'];
```

[1] Если вам не нужно искать в Google, что такое QBert, то можете всплакнуть по своей ушедшей юности.

```
$facebook_url = $_REQUEST['facebook_url'];
$twitter_handle = $_REQUEST['twitter_handle'];
```

Две из них взаимосвязаны: `$first_name` и `$last_name`. Информация о том, как зовут пользователя, довольно часто делится на имя и фамилию, но при этом она *редко* выводится по отдельности. Представьте, что вы заходите в мебельный магазин и вас вдруг окликает старый приятель со словами: «Привет, человек по имени Бретт и по фамилии Маклафлин!» Это прозвучало бы весьма неуклюже, и точно так же неуклюже это выглядит при общении в Интернете.

Поскольку мириться с таким разделением нам ни к чему, эти две строки можно просто объединить, воспользовавшись так называемой *конкатенацией*. Это непривычное слово просто означает объединение, и если, в частности, речь идет о строках, оно говорит об объединении частей текста. Таким образом, при конкатенации my и girl вы получите новую строку mygirl.

В PHP конкатенация осуществляется с помощью точки (`.`). Найдите в `getFormInfo.php` две строки HTML-кода, где выводятся имя и фамилия:

```
Имя: <?php echo $first_name; ?><br />
Фамилия: <?php echo $last_name; ?><br />
```

Теперь сведите их в одну строку и объедините имя и фамилию:

```
Имя: <?php echo $first_name . $last_name; ?><br />
```

Вернитесь к `socialEntryForm.html`, введите какую-нибудь информацию и отправьте форму. Результат показан на рис. 2.3.

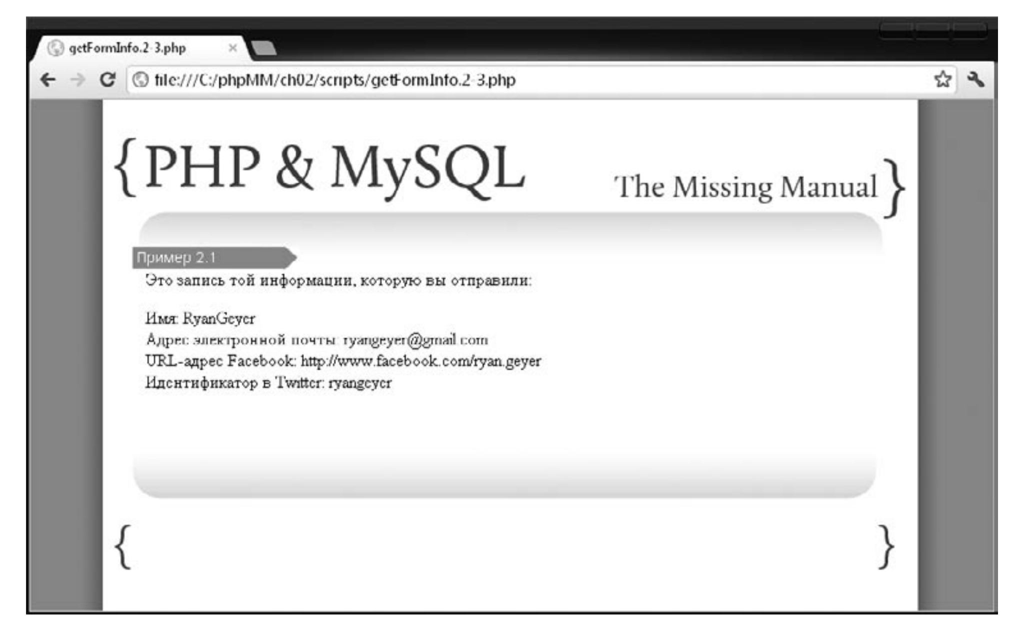


Рис. 2.3. Такие простые вещи, как объединение имени и фамилии, существенно улучшают восприятие веб-формы

Все получились! Но выявилась одна проблема: имя и фамилия написаны слитно. Нужно между этими двумя частями текста поставить пробел.

Здесь и проявляется польза от того, что PHP рассматривает весь текст одинаково. Пробел можно добавить просто поместив его в кавычки: " ". PHP не увидит, что этот текст чем-то отличается от текста в переменных. Поэтому вы можете просто объединить эту строку — пустое пространство — с $first_name, а затем с $last_name:

```
Имя: <?php echo $first_name . " " . $last_name; ?><br />
```

Попробуйте еще раз вывести данные формы, и вы увидите между именем и фамилией нужный нам пробел. Проверьте, соответствует ли изображение на рис. 2.4 теперешнему внешнему виду вашей страницы.

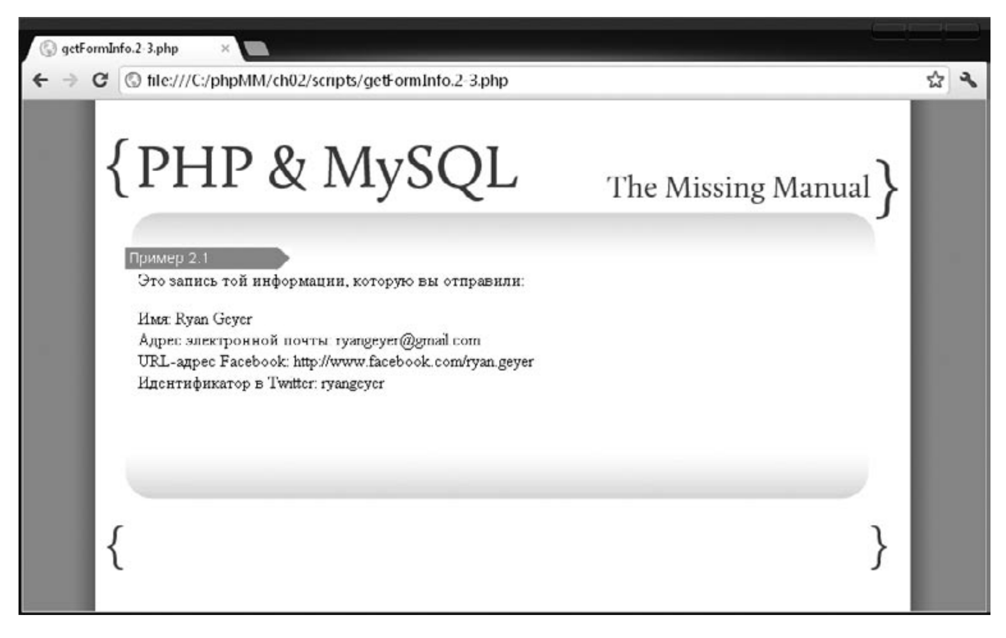


Рис. 2.4. PHP все равно, где находится текст: в переменной вроде $_REQUEST, в переменной, созданной вами, или в кавычках; он рассматривает все варианты текста одинаково

Поиск в тексте

Если строки можно было всего лишь объединять, с ними было бы скучно работать. Но PHP предоставляет намного больше возможностей. В PHP одним из наиболее распространенных действий с текстом считается поиск. Возьмем, к примеру, имеющуюся у нас переменную $facebook_url. Предположим, нужно превратить ее в настоящую ссылку, на которой можно будет щелкать кнопкой мыши:

```
<p>
  Имя: <?php echo $first_name . " " . $last_name; ?><br />
  Адрес электронной почты: <?php echo $email; ?><br />
```

```
  <a href="<?php echo $facebook_url; ?>">URL-адрес в Facebook:</a><br />
  Идентификатор в Twitter: <?php echo $twitter_handle; ?><br />
</p>
```

Теперь вместо простой демонстрации текста URL-адрес превратился в настоящую ссылку, на которой можно щелкать (рис. 2.5).

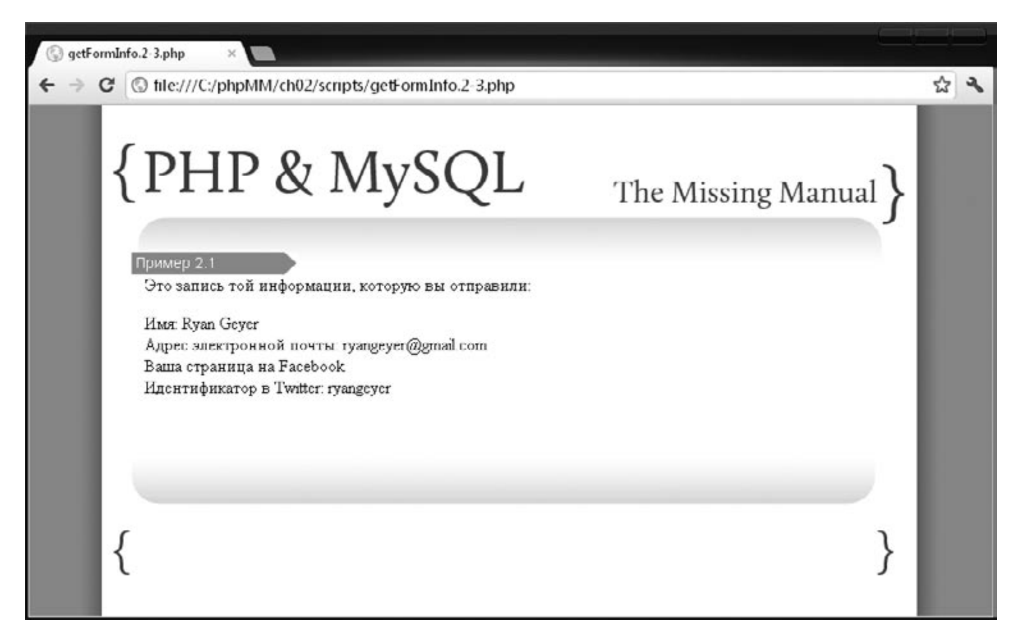


Рис. 2.5. Получая URL или адрес электронной почты, вы можете попробовать превратить их там, где это возможно, в ссылки HTML

А если кто-нибудь забудет поместить в URL-адрес часть facebook.com? Возможно, он невнимательно прочел, что от него требуется, и ввел в поле только ту часть URL-адреса, которая следует за facebook.com, например ryan.geyer или profile.php?id=699186223. Тогда создаваемая вами ссылка будет нерабочей.

Поэтому нужно посмотреть, содержит ли текст полученной нами переменной $facebook_url фрагмент facebook.com. Если содержит, то, наверное, мы можем спокойно превратить текст в URL-ссылку. Но если его там нет, то к началу значения переменной следует добавить текст http://www.facebook.com.

В PHP это проще всего сделать путем определения позиции части текста внутри более крупного текста. Таким образом можно увидеть, какую позицию занимает facebook.com внутри $facebook_url:

```
$first_name = $_REQUEST['first_name'];
$last_name = $_REQUEST['last_name'];
$email = $_REQUEST['email'];
$facebook_url = $_REQUEST['facebook_url'];
$position = strpos($facebook_url, "facebook.com");
$twitter_handle = $_REQUEST['twitter_handle'];
```

Функция strpos() (ее имя означает string position (позиция строки)) возвращает число, сообщающее о том, где в строке, в которой ведется поиск, находится искомая строка. Поэтому если значение переменной $position было равно 5, значит facebook.com внутри значения переменной $facebook_url располагается в позиции 5. (Если вы удивлены тем, почему 5, а не 6, смотрите следующую врезку.)

Но просто получить позицию недостаточно. С ней еще нужно что-то сделать. Необходимо определить, находится ли эта позиция внутри значения переменной $facebook_url (то есть $facebook_url содержит **facebook.com**) или же в значении $facebook_url вообще нет facebook.com. Это можно сделать, посмотрев, не является ли значение переменной $position ложным (не равно ли оно false). В противном случае strpos() возвращает позицию внутри значения переменной $facebook_url, в которой появляется искомая строка.

ПРИМЕЧАНИЕ

Функция strpos(), как и многие другие функции в PHP, может вернуть два совершенно разных значения: число, показывающее позицию внутри строки, в которой ведется поиск, или же значение false.

```
$first_name = $_REQUEST['first_name'];
$last_name = $_REQUEST['last_name'];
$email = $_REQUEST['email'];
$facebook_url = $_REQUEST['facebook_url'];
$position = strpos($facebook_url, "facebook.com");
if ($position === false) {
  $facebook_url = "http://www.facebook.com/" . $facebook_url;
}
$twitter_handle = $_REQUEST['twitter_handle'];
```

На первый взгляд все, возможно, выглядит так, будто в коде появилось много нового, но не стоит беспокоиться. Вы уже можете понять практически все, что там есть.

1. Сначала используется функция strpos(), чтобы посмотреть, имеется ли в значении переменной $facebook_url фрагмент facebook.com. Значение, возвращенное функцией strpos(), помещается в новую переменную $position.
2. Значение переменной $position сравнивается со специальным PHP-значением false, используемым в инструкции if. Вскоре вы познакомитесь с инструкцией if более подробно, но она всего лишь делает нечто похожее на следующее: если $position имеет значение false, она выполняет код, находящийся внутри фигурных скобок { }.
3. Код, который находится внутри { }, выполняется только в том случае, если инструкция, находящаяся над ним, вычисляется в true. В нашем случае это происходит, если $position === false. Если это условие является истиной, то к значению $facebook_url добавляется http://www.facebook.com, чтобы получилась нормальная ссылка на Facebook.
4. В коде скрыт еще один шаг: если $position не имеет значение false, то ничего не происходит. Строка кода внутри { } просто полностью обходится.

ПОД КАПОТОМ

Языкам программирования нравится начинать все с нуля

Чем больше приходится программировать на таких языках, как PHP, Java, C или Perl, тем больше попадаются случаи непривычного использования числа 0. Почти во всех этих языках (в PHP тоже) счет ведется с нуля, а не с единицы. Поэтому, например, если идет подсчет длины строки текста That's weird, то первая буква — заглавная «T» — будет находиться в позиции 0, а не 1.

Такая система проявляет особое коварство при поиске текста внутри текста. Предположим, что кто-то набрал в текстовом поле URL-адрес в Facebook нашей программы `getFormInfo.php` следующий текст: `facebook.com/michael.greenfield`. А затем вы в своем коде сделали нечто подобное, чтобы увидеть, не является ли значение поля формы настоящим URL:

```
if (strpos($facebook_url,
                 "facebook.com") > 0) {
   $facebook_url =
         "http://www.facebook.com/" .
          $facebook_url;
}
```

На первый взгляд PHP-код выглядит вполне прилично: если `facebook.com` не появляется в первой или в более высокой позиции значения `$facebook_url`, то программа должна добавить `http://www.facebook.com/` впереди значения `$facebook_url`.

Но результат вас не обрадует, поскольку у переменной `$facebook_url` будет значение, похожее на следующее: `http://www.facebook.com/facebook.com/michael.greenfield`. Почему так получилось?

Как я уже сказал выше, счет в PHP начинается с 0, а не с 1. Поэтому позиция 0 на самом деле является первой позицией в значении `$facebook_url`. И в этой позиции находится буква `f`. В позиции 1 находится буква `a`, в позиции 2 — буква `c` и т. д. Получается, что целиком первая часть значения `$facebook_url` содержит `facebook.com`, то есть строку, которую ищет ваш код. Поэтому `strpos()` вернет 0, чтобы показать, что искомая строка располагается в первой позиции значения переменной `$facebook_url`.

Все это означает (помимо того факта, что языки программирования ведут подсчет иначе, чем люди), что при написании кода вы должны переключать мышление на счет, начинающийся с нуля. Поэтому при поиске в строке позиция, означающая, что строка найдена, начинается не с 1 и более, а с 0 и более. Если помнить об этом, то можно уберечься от длительного поиска ошибок.

Теперь, когда вы внесли изменения в свой сценарий, сохраните его и вернитесь к веб-форме `socialEntryForm.html`. На это раз введите ссылку на Facebook, не набирая в ней часть URL-адреса `facebook.com`, например, введите только `profile.php?id=100000039185327`. Затем отправьте свою форму и посмотрите на результат.

На первый взгляд ничего не изменилось. Веб-страница, сгенерированная вашим PHP-интерпретатором, похожа на ту, что изображена на рис. 2.5. Но взгляните

на исходный код страницы (показанный на рис. 2.6) или щелкните на самой ссылке. В обоих случаях вы убедитесь в том, что текст `profile.php?id=100000039185327` превратился в настоящий URL-адрес `http://www.facebook.com/profile.php?id=100000039185327` (рис. 2.7).

getFormInfo.2-3.php — view-source:file:///C:/phpMM/ch02/scripts/getFormInfo.2-3.php

```
<html>
 <head>
  <link href="../../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пример 2.1</div>

  <div id="content">
    <p>Это запись той информации, которую вы отправили:</p>
    <p>
      Имя: Ryan Geyer<br />
      Адрес электронной почты: ryangeyer@gmail.com<br />
      <a href="http://www.facebook.com/profile.php?id=100000039185327">Ваша страница на Facebook</a><br />
      Идентификатор в Twitter: ryangeyer<br />
    </p>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

Рис. 2.6. Просматривать исходный код ваших веб-страниц — очень полезная привычка

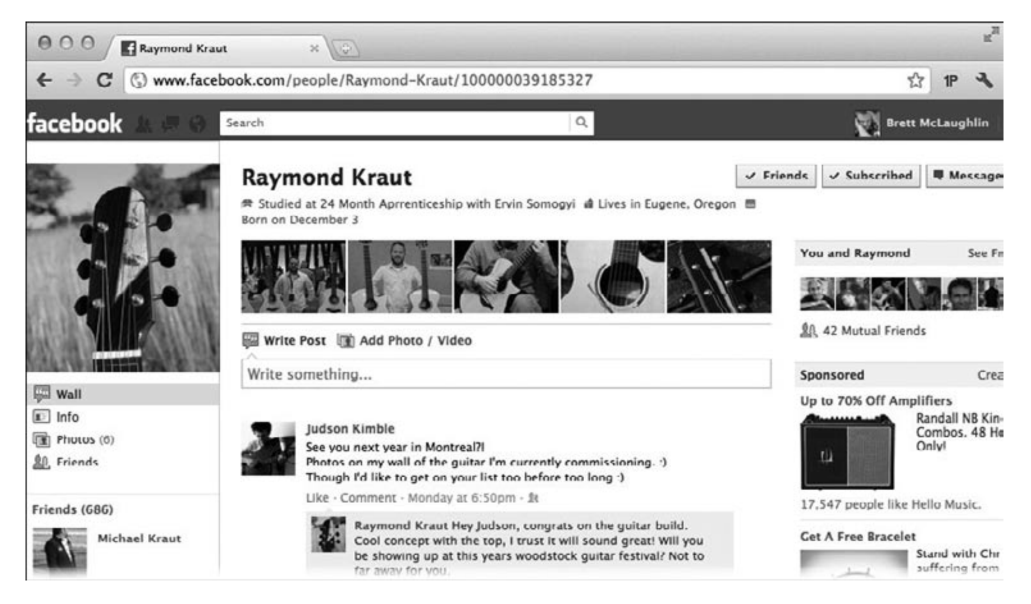


Рис. 2.7. Превращение неполного URL-адреса в работоспособную ссылку — трудоемкое, но важное дело

Изменение текста

Вы объединили два текстовых фрагмента, провели поиск внутри текста, что еще осталось неосвоенным? Ну, разумеется, изменение текста. У нас уже полно мест, требующих изменения текста.

Возьмем, к примеру, идентификатор Twitter, введенный в веб-форму. Большинство людей ставят перед своим идентификатором Twitter символ @, придавая ему следующий вид: @bdmclaughlin. Но для просмотра на сайте www.twitter.com чьего-нибудь профиля символ @ не нужен. Если, например, идентификатор Twitter имеет значение @phpGuy, то URL-адрес Twitter для просмотра профиля будет иметь значение http://www.twitter.com/phpGuy.

Следовательно, чтобы превратить идентификатор Twitter в рабочую ссылку, нужно выполнить следующие действия.

1. Создать новую переменную `$twitter_url` и присвоить ей начальное значение `http://www.twitter.com/`.
2. Определить, есть ли в идентификаторе Twitter символ `@`.
3. Если символа `@` в `$twitter_handle` нет, просто добавить идентификатор в качестве окончания значения `$twitter_url`.
4. Если символ `@` в `$twitter_handle` присутствует, удалить `@` из идентификатора и добавить этот идентификатор в качестве окончания значения `$twitter_url`.
5. Вывести идентификатор Twitter в качестве части элемента ссылки `<a>` в HTML-выводе сценария.

Нечто подобное всем этим действиям, за исключением четвертого, вы уже совершали, поэтому написание данной программы не превратится для вас в большую проблему.

Сначала создайте новую переменную для хранения выстраиваемого URL-адреса Twitter и присвойте ему в качестве значения первую часть этого адреса:

```
$twitter_handle = $_REQUEST['twitter_handle'];
$twitter_url = "http://www.twitter.com/";
```

Затем нужно определить, есть ли где-нибудь в идентификаторе Twitter, полученном в переменной `$twitter_handle`, символ `@`. Для этого опять можно воспользоваться функцией `strpos()`:

```
$twitter_handle = $_REQUEST['twitter_handle'];
$twitter_url = "http://www.twitter.com/";
$position = strpos($twitter_handle, "@");
```

В данном случае что-то нужно делать в зависимости от наличия или отсутствия символа `@` в `$twitter_handle`. Поэтому будет использоваться не только `if`, но и `else`:

```
$twitter_handle = $_REQUEST['twitter_handle'];
$twitter_url = "http://www.twitter.com/";
```

```
$position = strpos($twitter_handle, "@");
if ($position === false) {
  $twitter_url = $twitter_url . $twitter_handle;
} else {
  // некие действия по удалению символа @ из идентификатора Twitter
}
```

В этот код должен быть заложен вполне определенный смысл. Если символ @ отсутствует, нужно просто добавить идентификатор к концу значения переменной `$twitter_url`. Если символ @ присутствует, необходимо проделать дополнительную работу.

Вы уже видели, что функции `strpos()` передается строка, а затем еще одна строка, которую нужно найти. В PHP имеется похожий способ получения только части строки: `substr()`. Название `substr()` является сокращением слова substring (подстрока), что просто означает часть строки. Функции `substr()` передается строка, а затем начальная позиция.

Таким образом, выражение `substr("Hello", 2)` вернет вам строку `"llo"`. Такой результат обусловлен тем, что `"H"` находится в позиции 0, `"e"` — в позиции 1, а первая буква `"l"` — в позиции 2. Поскольку функции `substr()` в качестве стартовой позиции передано число 2, вы получаете буквы от этой позиции и до конца строки: `"llo"`.

ВНИМАНИЕ

Следует запомнить, что большинство PHP-функций, таких как substr() и strpos(), начинают счет с нуля. Если вы еще не усвоили, как они работают, вернитесь к врезке «Под капотом. Языкам программирования нравится начинать все с нуля».

В случае с идентификатором Twitter функцию `substr()` можно применить точно так же. Но вам нужно обрезать все с начала строки, включая символ @, чья позиция, как вы уже знаете, хранится с переменной `$position`. Поэтому можно воспользоваться функцией `substr()` и выбрать начало новой строки с позиции, следующей сразу за значением переменной `$position`, то есть `$position + 1`.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Что творится со всеми этими угловыми скобками?

Когда PHP используется для вывода большого объема HTML и когда небольшие фрагменты PHP включаются в этот HTML, нетрудно и запутаться. Рассмотрим одну из строк, имеющихся в программе `getFormInfo.php`:

```
<a href="<?php echo $facebook_url; ?>">
  Ваша страничка на Facebook
</a><br />
```

Здесь, мягко говоря, есть одна странность: имеются две открывающие угловые скобки в начале, а затем две закрывающие угловые скобки в конце строки. И, кроме того, весь PHP-код заключен в кавычки.

К сожалению, такие странности являются одним из недостатков вставки PHP в HTML. С этим приходится мириться, и к этому нужно привыкать. PHP-код должен быть заключен между символами <?php и ?>. (Это не строгое правило, и если сценарий завершается кодом PHP, то символы ?> можно не ставить, но считается, что такая практика — удел ленивых.) И в том случае, когда PHP используется для вставки чего-нибудь в элемент, который уже находится в угловых скобках, получается такой вот странный код, имеющий двойные угловые скобки.

Кроме того, PHP часто используется для создания ссылок, которые в случае применения элемента a становятся значением его атрибута. Следовательно, весь PHP-блок будет взят в кавычки.

Пока ваш PHP сам не имеет кавычек, это вполне приемлемо. Когда же вы сталкиваетесь с необходимостью поставить кавычки внутри PHP-кода, который уже взят в кавычки, можно чередовать одинарные и двойные кавычки:

```
<a href="<?php echo
         'http://www.twitter.com/' .
          $twitter_handle; ?>">
  Ваша страничка на Facebook
</a><br />
```

Можно также без всяких проблем поменять одинарные и двойные кавычки местами:

```
<a href='<?php echo
          "http://www.twitter.com/" .
           $twitter_handle; ?>'>
  Ваша страничка на Facebook
</a><br />
```

Нужно лишь убедиться в том, что вы не открываете какой-нибудь фрагмент с помощью одинарных кавычек и не закрываете его двойными кавычками или наоборот. Это нарушит порядок вещей, с чем никому не хочется столкнуться.

Вообще-то, в порядке обработки строк в двойных и одинарных кавычках в PHP существуют некоторые различия, но в данном случае они нас не касаются.

```
$twitter_handle = $_REQUEST['twitter_handle'];
$twitter_url = "http://www.twitter.com/";
$position = strpos($twitter_handle, "@");
if ($position === false) {
  $twitter_url = $twitter_url . $twitter_handle;
} else {
  $twitter_url = $twitter_url . substr($twitter_handle, $position + 1);
}
```

ПРИМЕЧАНИЕ

Вы приступаете к беглому просмотру больших фрагментов нового кода, но не стоит волноваться, если на первый взгляд что-то будет непонятно. Просто изучите каждый отдельный фрагмент нового кода. Это очень быстро прояснит общую картину.

Теперь осталось только обновить ту часть сценария, которая выводит HTML:

```
<p>
  Имя: <?php echo $first_name . " " . $last_name; ?><br />
  Адрес электронной почты: <?php echo $email; ?><br />
  <a href="<?php echo $facebook_url; ?>">URL-адрес в Facebook</a><br />
  <a href="<?php echo $twitter_url; ?>">Проверьте свои записи в Twitter</a><br />
</p>
```

Перейдите назад на страницу ввода данных, заполните ее информацией, а затем отправьте форму обновленному сценарию. Испытайте работу при наличии символа @ в идентификаторе Twitter и при его отсутствии. Результат при этом должен быть одинаковым: красивая страничка вывода со ссылками на ваши страницы Facebook и Twitter с корректно удаленным символом @ (рис. 2.8).

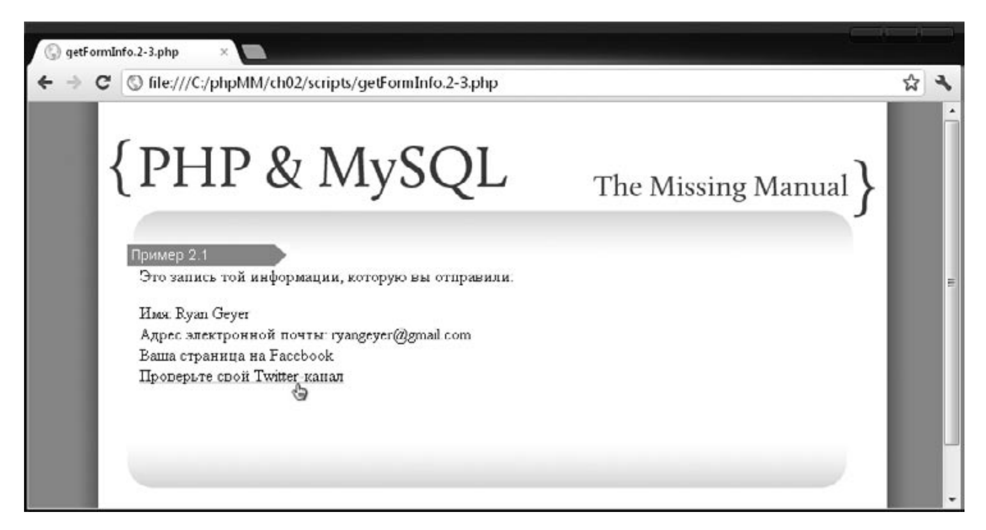


Рис. 2.8. Результат действия программы

Данному сценарию PHP можно добавить стилевое оформление и форматирование. У вас может появиться желание изменить надписи, чтобы они читались от первого лица: «Мое имя» и «Проверка моих записей в Twitter». Теперь, когда вы уже достаточно освоили свой PHP-сценарий, не бойтесь экспериментировать.

Обрезка и замена текста

При попытках оказать пользователям помощь при возможных ошибках ввода информации в формы работа со строками в PHP превращается в обширный инструментарий, предоставленный к вашим услугам. Рассмотрим еще две весьма распространенные проблемы, возникающие при заполнении веб-форм, особенно тех, в которые пользователи вводят URL-адреса:

- пользователи вводят ненужные пробелы, например вместо «`http://www.facebook.com/ryan.geyer`» набирают « `http://www.facebook.com/ryan.geyer   `»;

- пользователи путают домены com и org URL-адресов, вводя вместо «http://www.facebook.com/profile.php?id=534643138» строку вроде «http://www.facebook.org/profile.php?id=534643138».

ПРИМЕЧАНИЕ

Как ни удивительно, люди довольно часто путают com и org. Фактически многие компании, владеющие доменным именем «имя_домена.com», покупают также доменное имя «имя_домена.org» и перенаправляют все обращения к «имя_домена.org» на адрес «имя_домена.com» из-за распространенного характера данной проблемы.

Вы уже знаете, как работать со строками в PHP, и уже пользовались некоторыми PHP-функциями, поэтому, чтобы справиться с этими общими проблемами, нужно просто изучить еще две функции.

Удаление лишних пробелов с помощью функции trim()

PHP-функция trim() удаляет в начале и конце строки любые пустые места, то, что в PHP называется *пробельными символами*. То есть обрезка строки « I love my space bar. » даст в результате строку «I love my space bar.».

ПРИМЕЧАНИЕ

В PHP есть также функция rtrim(), обрезающая только пробелы, которые находятся в конце строки (в ее правой части), и функция ltrim(), обрезающая пробелы в начале строки (в ее левой части).

Итак, с помощью всего лишь пары простых добавлений к сценарию можно гарантировать, что лишние пробелы в начале и конце строки с введенными пользователем данными уйдут в прошлое:

```
$first_name = trim($_REQUEST['first_name']);
$last_name = trim($_REQUEST['last_name']);
$email = trim($_REQUEST['email']);
$facebook_url = trim($_REQUEST['facebook_url']);
$position = strpos($facebook_url, "facebook.com");
if ($position === false) {
  $facebook_url = "http://www.facebook.com/" . $facebook_url;
}

$twitter_handle = trim($_REQUEST['twitter_handle']);
$twitter_url = "http://www.twitter.com/";
$position = strpos($twitter_handle, "@");
if ($position === false) {
  $twitter_url = $twitter_url . $twitter_handle;
} else {
  $twitter_url = $twitter_url . substr($twitter_handle, $position + 1);
}
```

Изменения весьма просты: при каждом получении значения из переменной $_REQUEST его нужно просто заключить в функцию trim(). И теперь больше уже не нужно беспокоиться насчет лишних пробелов в начале или конце строки текста.

ВНИМАНИЕ

Функция trim() (впрочем, как и функции rtrim() и ltrim()) удаляет только те пробелы, которые находятся за пределами текста. Поэтому функция trim() хорошо подходит для работы с такими строками, как « Ого, как много пробелов. », но не поможет при удалении лишних пробелов в таких строках, как «Ого,   как много пробелов.»

Замена символов в тексте с помощью функции str_replace()

Текст в строке также несложно заменить. Для этого используется функция `str_replace()`, которой передаются три аргумента.

- Искомый текст. Если вы ищете `facebook.org`, искомым текстом будет `"facebook.org"`.
- Текст замены. Если каждое появление `facebook.org` заменяется `facebook.com`, то текст замены будет таким — `"facebook.com"`.
- Строка, в которой будет вестись поиск. В данном случае — значение, введенное пользователем в веб-форму.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Формируйте цепочку методов (или не делайте этого!)

Во многих случаях все, что можно сделать по отдельности, PHP позволяет сделать за один шаг. Рассмотрим следующий фрагмент PHP-сценария:

```
$facebook_url =
  str_replace(
    "facebook.org",
    "facebook.com",
    trim($_REQUEST['facebook_url']));
```

В этом коде объединены несколько разных действий. Его можно переписать следующим образом, чтобы сделать отдельные действия немного понятнее:

```
$facebook_url = $_REQUEST['facebook_url'];
$facebook_url = trim($facebook_url);
$facebook_url =
  str_replace(
    "facebook.org", "facebook.com",
    $facebook_url);
```

Оба этих кодовых фрагмента работают одинаково, и с точки зрения производительности и техники ни один из них не имеет заметных преимуществ. Стало быть, выбор за вами. И каково же будет ваше решение?

Есть два основных решения этого вопроса. Первое из них распространен в программистских кругах, и его суть выражается в подходе к программированию с девизом «Краткость — сестра таланта»: зачем что-то делать в двух, трех или четырех строках, когда все это можно сделать в одной строке? Следовательно, с таким подходом

нужно пользоваться любой возможностью объединения шагов. Код становится короче, отпадает необходимость в большом количестве промежуточных шагов. В результате получается то, что называется *формированием цепочки методов*. Например, с текстовым фрагментом делается что-то одно, а затем результат этого одного действия отправляется другому действию. Иными словами, каждый шаг — звено цепи, и вся строка является завершенной и готовой к использованию цепью.

Другое решение менее популярно среди программистов, если только эти программисты не должны обучать своему ремеслу кого-нибудь другого. Превыше всего в этом решении — упростить понимание кода. Следовательно, чем больше удастся разорвать цепочку действий, тем проще будет разобраться в происходящем. Для этого требуется набирать больше кода, но весь дополнительный код служит для облегчения понимания и (по крайней мере в теории) устранения причин неверного поведения кода.

Правильнее всего будет, наверное, найти некую золотую середину между этими двумя подходами. Например, ваш код в программе getFormInfo.php пока еще вполне понятен, хотя некоторые действия там объединены в цепочку. Но если в одной строке кода сводятся воедино 6, 7 и даже 10 действий, то на их разделение может понадобиться время.

Соберем все это воедино, и у нас получится нечто подобное следующему коду:

```
$facebook_url = str_replace("facebook.org", "facebook.com",
                            trim($_REQUEST['facebook_url']));
$position = strpos($facebook_url, "facebook.com");
if ($position === false) {
  $facebook_url = "http://www.facebook.com/" . $facebook_url;
}
```

ПРИМЕЧАНИЕ

Дополнительные сведения о том, почему функция str_replace() имеет такой вид, изложены в предыдущей врезке.

Внесите эти изменения, а затем снова зайдите в вашу веб-форму. Введите информацию, которая могла бы вызвать проблемы у менее искушенного PHP-программиста, — с массой пробелов и неправильным URL-адресом facebook.org, например, как на рис. 2.9.

Отправьте эти данные, и увидите картину, показанную на рис. 2.10, свидетельствующую о том, что программа getFormInfo.php не пропустила удар. Она избавилась от всех лишних пробелов и даже исправила неверный URL-адрес.

Обратите еще внимание на режим View Source (Просмотр кода страницы). Это ваш помощник. В большинстве браузеров этот пункт присутствует в меню просмотра, меню страницы или же доступен после щелчка правой кнопкой мыши

Рис. 2.9. Ввод такой информации мог бы вызвать проблемы

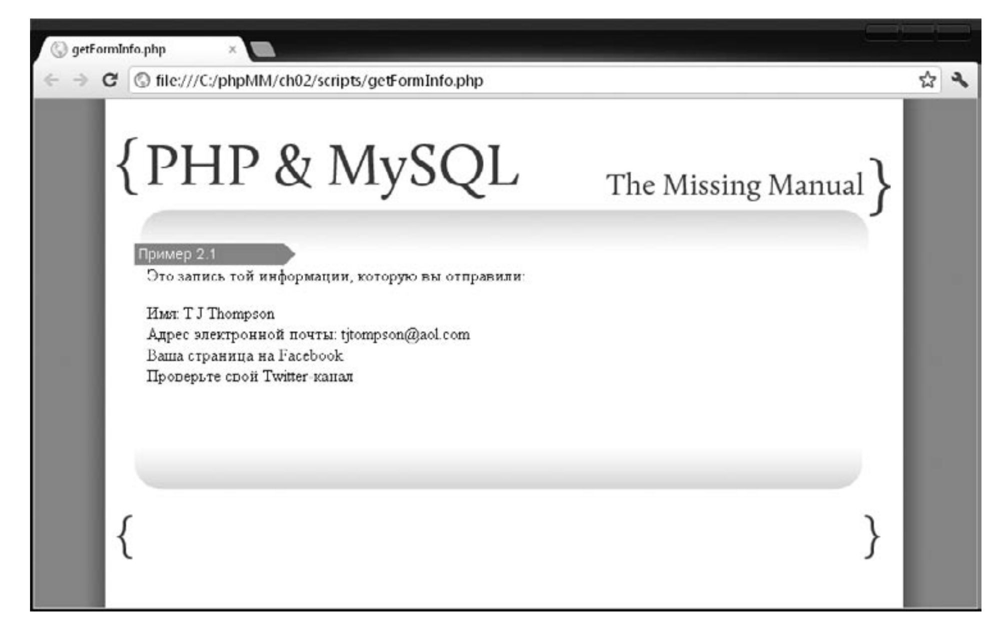


Рис. 2.10. Результат ввода данных

на странице. Убедитесь в том, что вы можете просматривать исходный код вашей страницы. Именно он и отправляется браузеру, независимо от того, что и как выглядит на вашей странице (рис. 2.11).

```
view source:file:///C:/phpMM/ch02/scripts/getFormInfo.php
<html>
 <head>
  <link href="../../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пример 2.1</div>

  <div id="content">
    <p>Это запись той информации, которую вы отправили:</p>
    <p>
      Имя: T J Thompson<br />
      Адрес электронной почты: tjtompson@aol.com<br />
      <a href="http://www.facebook.com/profile.php?id=534678021">Ваша страница на Facebook</a><br />
      Проверьте свой Twitter-канал<br />
    </p>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

Рис. 2.11. Код программы в режиме View Source (Просмотр кода страницы)

Переменная $_REQUEST

PHP — не просто способ работы с текстом, он предоставляет значительно более широкие возможности. Выше мы рассматривали строки, но существует намного больше типов данных, с которыми приходится работать в PHP-сценариях. Нетрудно, например, предположить, что существуют всевозможные способы работы с числами, и вскоре мы с ними столкнемся.

Есть еще и другой не менее важный тип данных, который нужно освоить. Фактически вы уже встречались с ним, поскольку работали с текстом. Этим таинственным типом является *массив*: разновидность контейнера, который в действительности содержит внутри себя другие значения.

ВОЗЬМИТЕ НА ЗАМЕТКУ

PHP предлагает массу функций для работы со строками

Мы всего лишь прошлись по верхам того, что PHP предлагает для работы со строками и текстом. Вы можете посетить страницу www.php.net/manual/en/ref.strings.php, чтобы получить полный перечень всего, что можно делать с текстом в PHP.

Данный список очень длинный. Но не стоит волноваться: вам не придется запоминать все эти функции. Просто сохраните закладку на эту страницу и на руководство по PHP на странице www.php.net/manual и помните, что при необходимости вы можете найти там описание нужных функций. Если вам попадается строка, с которой следует работать, просто откройте по закладке руководство по PHP и поищите в нем необходимую функцию. Так поступают все программисты. (Конечно, попадаются уникумы, подобные персонажу Дастина Хоффмана, способные отбарабанить все имеющиеся в PHP функции, эдакие люди дождя в сером костюме. Но эти люди — исключение из правил.)

⇨

Вместо того чтобы стараться запомнить все ненужные подробности каждой функции, имеющейся в языке PHP, разберитесь с шаблонами PHP. Например, теперь вы уже знаете, что часто при работе со строками требуется вызов неких функций, передача им информационных фрагментов и присваивание результата переменной. Именно это важно знать. Теперь всякий раз при поиске функции работы со строками в руководстве по PHP, вы будете уверены в том, как правильно использовать эту функцию.

Массивы могут содержать несколько значений

Массив является типом *структуры данных*. Это одно из тех понятий, которое придаст вам вес на программистских форумах, но если у вас будет мешанина в голове, вы будете выглядеть довольно странно. На самом деле в массивах нетрудно разобраться. Представьте, например, массив в виде картотеки данных.

Итак, если у вас есть переменная по имени, скажем, `$file_cabinet` и она является массивом, значит, она может хранить внутри себя другую информацию. Переменную `$file_cabinet` можно наполнить URL-адресами, именами и фамилиями, адресами электронной почты и т. д. Заполнение картотеки производится путем сообщения PHP с помощью чисел и квадратных скобок, которые ставятся сразу после имени переменной массива, информации о том, где должны находиться ваши данные:

```
<?php

$file_cabinet[0] = "Derek";
$file_cabinet[1] = "Trucks";
$file_cabinet[2] = "derek@DerekTrucks.com";
$file_cabinet[3] = "http://www.facebook.com/DerekTrucks";
$file_cabinet[4] = "@derekandsusan";

?>
```

Эти числа подобны выдвижным ящикам картотеки, или, если вы предпочитаете что-нибудь поминиатюрнее, ярлыкам на папках с файлами картотеки.

ПРИМЕЧАНИЕ

Когда вам будут попадаться подобные примеры кода, вы можете набирать их на компьютере, сохранять на диске (используя имена с расширением PHP) и запускать с помощью команды php. В таком случае вы сможете лучше во всем разобраться и быстро научиться создавать собственные программы.

Затем информацию из `$file_cabinet` можно извлечь, использовав те же самые числа в квадратных скобках:

```
$first_name = $file_cabinet[0];
$last_name = $file_cabinet[1];
$email = $file_cabinet[2];
$facebook_url = $file_cabinet[3];
$twitter_handle = $file_cabinet[4];
```

ВНИМАНИЕ

Возможно, для вас это уже и не новость, но стоит напомнить, что в PHP в большинстве случаев счет начинается с нуля (см. врезку «Под капотом. Языкам программирования нравится начинать все с нуля» в разделе «Поиск в тексте» данной главы). Массивы в этом случае не исключение. Поэтому первый элемент в $file_cabinet хранится в $file_cabinet[0], а не в $file_cabinet[1].

После этого вы сможете делать с этими значениями все, что угодно, включая их вывод на экран. Рассмотрим вполне законченную программу, от которой, правда, мало пользы, но зато она проверяет массив в деле. Она заполняет массив, извлекает информацию из массива, а затем выводит ее на экран:

```
<?php

$file_cabinet[0] = "Derek";
$file_cabinet[1] = "Trucks";
$file_cabinet[2] = "derek@DerekTrucks.com";
$file_cabinet[3] = "http://www.facebook.com/DerekTrucks";
$file_cabinet[4] = "@derekandsusan";

$first_name = $file_cabinet[0];
$last_name = $file_cabinet[1];
$email = $file_cabinet[2];
$facebook_url = $file_cabinet[3];
$twitter_handle = $file_cabinet[4];

echo $first_name . " " . $last_name;
echo "\nАдрес электронной почты: " . $email;
echo "\nURL-адрес в Facebook: " . $facebook_url;
echo "\nИдентификатор в Twitter: " . $twitter_url;
?>
```

Теперь подумаем о пользе массивов. Кто-нибудь разве откажется от того, чтобы убрать нечто в папку, чтобы потом все это можно было легко найти? Но в данном случае есть одна проблема. Кто-нибудь сможет запомнить, что в позиции 2 можно получить фамилию, а в позиции 4 — URL-адрес Facebook? Со временем во всем этом можно будет окончательно запутаться.

К счастью, мудрые люди, создавшие PHP, продумали и этот вопрос. Массивы в PHP являются ассоциативными, а это значит, что они могут связывать ярлыки с каждым элементом массива. Возвращаясь к идее, что каждое число является папкой в картотеке, вы можете использовать текущий ярлык, наклеенный на папку. Причем этот ярлык может быть чем угодно.

Итак, рассмотрим ту же незатейливую программу, но на этот раз используем в ней ассоциативные ярлыки. Если вы проверяете все прочитанное на практике, можете внести в свою копию этого сценария следующие изменения:

```
<?php

$file_cabinet['first_name'] = "Derek";$file_cabinet['last_name'] = "Trucks";
$file_cabinet['email'] = "derek@DerekTrucks.com";
```

```
$file_cabinet['facebook_url'] = "http://www.facebook.com/DerekTrucks";
$file_cabinet['twitter_handle'] = "@derekandsusan";

$first_name = $file_cabinet['first_name'];
$last_name = $file_cabinet['last_name'];
$email = $file_cabinet['email'];
$facebook_url = $file_cabinet['facebook_url'];
$twitter_handle = $file_cabinet['twitter_handle'];

echo $first_name . " " . $last_name;
echo "\nАдрес электронной почты: " . $email;
echo "\nURL-адрес в Facebook: " . $facebook_url;
echo "\nИдентификатор в Twitter: " . $twitter_url;

?>
```

Теперь массив `$file_cabinet` выглядит более знакомым. Мы уже видели нечто очень похожее...

Работа с $_REQUEST как с массивом

Специальная переменная PHP, снабжающая вас всей информацией из веб-формы, называется `$_REQUEST`. Она также является массивом. Когда вы напишете код вида `$_REQUEST['first_name']`, вы просто извлечете конкретный фрагмент информации из этого массива.

Вы (или веб-браузер) помещаете информацию в массив, а затем извлекаете ее из массива и работаете с ней. И здесь неважно, что для всего этого применялся массив, он просто является удобным способом хранения данных, как в том случае, когда браузер отправляет запрос вашему PHP-сценарию.

Вы видели, что информацию из массива можно извлечь не только по имени, то есть по ярлыку на папке, но и по номеру. Поэтому для ее извлечения можно указать `$file_cabinet['first_name']` или `$file_cabinet[0]`. То же самое справедливо и для `$_REQUEST`, поскольку это всего лишь массив. Таким образом, указание `$_REQUEST[0]` в PHP вполне допустимо.

Итак, что же находится в `$_REQUEST`? Создадим новую программу, и вы все сами увидите:

```
<html>
 <head>
  <link href="../../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пример 2.2</div>

  <div id="content">
```

```
    <p>Это все, что записано в массиве $_REQUEST:</p>
    <?php
      foreach($_REQUEST as $value) {
        echo "<p>" . $value . "</p>";
      }
    ?>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

Это еще один сценарий, который поначалу может выглядеть устрашающе, но в нем нет ничего страшного. Фактически единственным, чего вы еще не видели, является строка с ключевым словом foreach (подробнее мы рассмотрим ее далее). А теперь взглянем на эту строку, с которой начинается цикл PHP:

```
foreach($_REQUEST as $value) {
```

foreach является в PHP ловкой выдумкой, позволяющей быстро получить значения массива. В данном случае foreach передается массив — $_REQUEST, а затем из этого массива поочередно выводятся все значения. Всякий раз, когда выводится отдельное значение, оно присваивается новой переменной по имени $value. Это та самая переменная $value, которая является частью строки foreach. Поэтому внутри цикла foreach имеется переменная $value, содержащая отдельное значение из массива.

Как и в инструкции if, уже использовавшейся несколько раз, фигурные скобки { } сообщают PHP о том, где начинается и заканчивается этот цикл:

```
foreach($_REQUEST as $value) {
  echo "<p>" . $value . "</p>";
}
```

Все, что находится между { }, запускается единожды при каждом проходе цикла. Это значит, что для каждого элемента, имеющегося в $_REQUEST, эта строка кода будет запущена один раз:

```
echo "<p>" . $value . "</p>";
```

Большой пользы эта строка не приносит: она просто выводит значение $value наряду с HTML-форматированием. Но при каждом проходе цикла в переменной $value содержится другое значение, извлеченное из $_REQUEST, по данной причине это быстрый способ вывести каждое значение, имеющееся в $_REQUEST.

Теперь предположим, что в $_REQUEST имеются значения вроде Derek, Trucks и @DerekAndSusan. Когда PHP запустит ваш код, то в конечном итоге он проделает нечто подобное:

```
echo "<p>" . "Derek" . "</p>";
echo "<p>" . "Trucks" . "</p>";
echo "<p>" . "@DerekAndSusan" . "</p>";
```

Сохраните рассмотренный сценарий в файле под именем showRequestInfo.php. Нужно также внести изменения в то место, куда веб-форма socialEntryForm.php отправляет свою информацию:

```
<form action="scripts/showRequestInfo.php" method="POST">
  <fieldset>
    <label for="first_name">Имя:</label>
    <input type="text" name="first_name" size="20" /><br />
    <label for="last_name">Фамилия:</label>
    <input type="text" name="last_name" size="20" /><br />
    <label for="email">Адрес электронной почты:</label>
    <input type="text" name="email" size="50" /><br />
    <label for="facebook_url">URL-адрес в Facebook:</label>
    <input type="text" name="facebook_url" size="50" /><br />
    <label for="twitter_handle">Идентификатор в Twitter:</label>
    <input type="text" name="twitter_handle" size="20" /><br />
  </fieldset>
  <br />
  <fieldset class="center">
    <input type="submit" value="Вступить в клуб" />
    <input type="reset" value="Очистить и начать все сначала" />
  </fieldset>
 </form>
```

ПРИМЕЧАНИЕ

У вас может появиться желание скопировать socialEntryForm.html и назвать его как-нибудь по-другому, например socialEntryForm-2.html или enterInformation.html. В таком случае у вас могут появиться две версии: одна, отправляющая информацию в showRequestInfo.php, и другая, которая посылает информацию в getFormInfo.php.

Теперь войдите в свою новую веб-форму, заполните ее и отправьте данные. В ответ будет получена весьма интересная веб-форма: результат запуска вашего нового сценария showRequestInfo.php. В конечном итоге вам будет сообщено, что на самом деле отправлялось между вашим веб-браузером и веб-сервером (рис. 2.12).

Итак, теперь у вас есть необработанная информация, но что все это означает? Эти результаты похожи на вид всех файлов на компьютере, но без имен этих файлов. Или, если нравится аналогия с картотекой, представьте выдвижной ящик с папками, у которых оторваны ярлыки. Это затрудняет понимание происходящего.

Что касается данных формы, то вы уже знаете ярлыки first_name, last_name, email и т. д. В ассоциативном массиве, подобном тому, что используется PHP, это называется *ключами*. Поэтому можно получить значение конкретной «папки» в массиве с помощью следующего кода:

```
$value = $file_cabinet[$key];
```

Этот код получает значение из массива, которое прикреплено к ярлыку, который содержится в переменной $key. Если значение $key было first_name, то код будет, по сути, аналогичен следующему:

```
$value = $file_cabinet['first_name'];
```



Рис. 2.12. Полученная веб-форма: ожидаемая информация и некие странные числа

По этой причине в showRequestInfo.php вам просто нужно также получить не только значения, но и ключи из массива $_REQUEST. Затем можно будет вывести и ключи, и значения. И если вы еще не в курсе, PHP с помощью foreach опять все упрощает:

```
<div id="content">
  <p> Это все, что записано в массиве $_REQUEST:</p>
  <?php
    foreach($_REQUEST as $key => $value) {
      echo "<p>Для " . $key . ", имеется значение '" . $value . "'.</p>";
    }
  ?>
</div>
```

На этот раз вы заставляете foreach получить ключ в виде значения $key и значение в виде $value. Специальный значок => сообщает PHP, что вам нужен ключ $key, а затем значение $value, прикрепленное к ключу. Иными словами, вы получаете ярлык и папку, к которой этот ярлык приклеен, то есть именно то, что нужно.

Еще раз заполните форму и проверьте результаты работы вашего обновленного сценария PHP (рис. 2.13). И наконец, раскрылся секрет этих загадочных чисел: они прикреплены к некоторым специальным ключам с названиями __utmz, __utma и __utmc. Кроме этого можно увидеть и ярлыки вашей веб-формы: first_name, last_name и т. д.

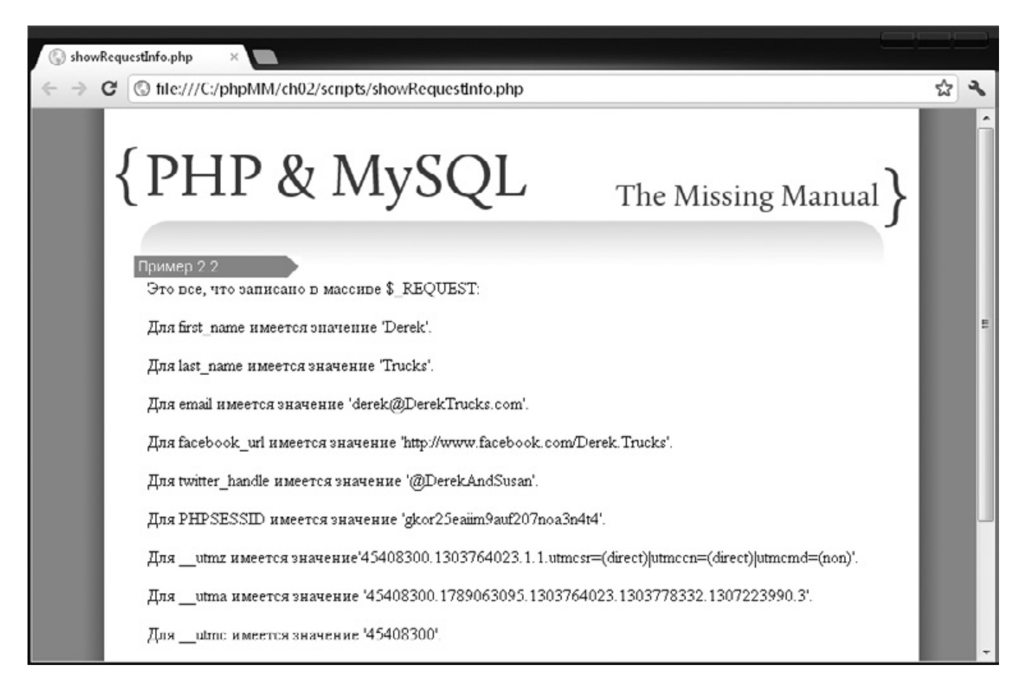


Рис. 2.13. Результат работы обновленного сценария PHP

ПРИМЕЧАНИЕ

Вы можете увидеть и нечто подобное тем загадочным элементам, которые хранятся в $_REQUEST: __utmz и __utmc, хотя это во многом зависит от веб-сервера и хостинг-провайдера. Это специальные HTTP-переменные, которые могут создаваться сервером. Вам не стоит о них беспокоиться.

Что делать с пользовательской информацией?

На данный момент вы располагаете некоторой информацией, помещенной в нескольких переменных. Фактически ваша ранняя веб-форма `socialEntryForm.html` выглядела во многом похожей на те формы подписки, которые вы, возможно, заполняли в сети сотни (если не тысячи!) раз. И тут, похоже, мы сталкиваемся с проблемой. Возможно, вы уже сталкивались с этим, если прорабатывали все изменения в вашем сценарии `getFormInfo.php`: ничего из этой информации никогда не сохранялось! Вам снова и снова приходилось вводить свое имя и социальную информацию.

Хорошие PHP-программисты способны решить практически любую поставленную перед ними техническую задачу. Им известны все PHP-функции для работы со строками, они знают все о массивах и многое другое. Но великолепные PHP-программисты могут решить полный набор проблем, о которых хорошие PHP-программисты даже не задумываются: это проблемы пользовательских ожиданий. Эти проблемы нельзя отнести к техническим, хотя, чтобы удовлетворить запросы пользователей, нужно, наверное, быть неплохим программистом.

Вопрос на миллион долларов: что ждет пользователь от ваших страниц и сценариев? Например, не ждет ли он, что придется вернуться на вашу страницу и ввести все ту же информацию снова и снова? Конечно же, нет. Вы ведь не стали бы возвращаться на подобный веб-сайт. Поэтому перед вами встает проблема ожиданий пользователя, и если вы хотите привлечь его к использованию вашего сайта, то эту проблему лучше решить.

Фактически наилучшим выходом для вас будет реально попользоваться своими собственными страницами и программами. Налейте чашку кофе, возьмите блокнот и сядьте за компьютер. Закройте все свои текстовые редакторы и программные средства и подумайте: «Я — пользователь!» Затем испытайте в деле свою веб-форму, отправьте ее, введите заведомо неправильную информацию и посмотрите, что из этого выйдет. Зафиксируйте в блокноте все, что вам не нравится, и помните: здесь вы выступаете в роли пользователя.

СОВЕТ

Возможно, у вас появится соблазн параллельно приступить к каким-то исправлениям. Не поддавайтесь искушению. Как только вы займетесь правкой или даже отвлечетесь на работу на компьютере, вы утратите мышление пользователя и что-нибудь упустите.

Наверное, у вас возникнут мысли, ранее никогда не приходившие вам в голову. Как ими распорядиться? Нужно приступать к исправлению возникшей ситуации. И в первую очередь нужно решить досадную проблему необходимости многократного ввода на вашей странице одной и той же информации.

3 MySQL и SQL: база данных и язык

Пожалуй, один из самых распространенных вопросов в мире: «Куда это делось?» Куда делась сахарница? Где мои ботинки? Куда делась коробка с новыми книгами? Куда делись квитанции? И поскольку подобный вопрос задается довольно часто, неудивительно, что при создании веб-приложений у вас тоже возникнет подобный вопрос: «Куда девается моя информация?»

Ответ по крайней мере для тех веб-приложений, которые создаются с помощью веб-страниц и PHP, довольно прост: «В базу данных». Да, база данных — это еще один инструмент, который нужно установить, а средство управления этой базой — еще один язык, который нужно изучить. Но, как вы увидите в этой главе, оно того стоит. Если вы создаете PHP-код, то вам также нужна база данных.

Что такое база данных?

База данных — это инструмент, позволяющий хранить информацию, получать эту информацию по мере надобности и систематизировать хранящуюся информацию. Разновидностью реальной базы данных является металлический картотечный шкаф. В него можно помещать документы, из него можно их извлекать, а также использовать папки и ярлыки для систематизации документов.

Базы данных являются постоянным хранилищем

Вы уже знаете, что в качестве картотеки PHP предоставляет вам массивы (см. подраздел «Массивы могут содержать несколько значений» раздела «Переменная $_REQUEST» главы 2). Но является ли массив базой данных? Он подпадает под определение в наипростейшем из возможных смыслов, но не предназначен для слишком длительного обслуживания ваших потребностей. Массивы и их содержимое в PHP становятся ненужным хламом при каждой остановке программы и ее повторном запуске. От такой базы данных толку мало. Уж лучше воспользоваться старым добрым металлическим картотечным шкафом.

Хорошая база данных может хранить информацию долгое время. Она не теряет информацию из-за того, что программа прекратила работу или был перезапущен весь веб-сервер. Представьте только, что при необходимости выключить веб-сервер для его обновления база данных потеряет имя, фамилию и адрес электронной почты каждого пользователя! Неужели вы думаете, что пользователи будут возвращаться на ваш сайт только для того, чтобы снова ввести все это? Конечно, нет.

Поэтому хорошая база данных должна быть более постоянным хранилищем. В понятиях программирования это называется сохранностью информации. Иными словами, если веб-сервер останавливается или даже если база данных должна быть остановлена и перезапущена, помещенная в базу данных информация остается на месте. (Вопрос о том, насколько долго она на самом деле остается, рассмотрен в следующей врезке.)

ПОД КАПОТОМ

Постоянные данные на самом деле являются полупостоянными

Несмотря на то что в базах данных хранится ваша информация и это хранилище выдерживает перезапуски компьютера или самой базы данных, информацию все же нельзя назвать постоянной. Подумайте вот о чем: даже если вы что-то записали ручкой, а не карандашом, который можно стереть, записку можно выбросить. Точно так же работают и базы данных: они сохраняют информацию в такой форме, которую трудно уничтожить, но саму информацию уничтожить можно.

Базы данных хранят информацию в каком-то месте, обычно на жестких дисках. Если один из этих жестких дисков сломается или на нем появятся нерабочие области, информация будет утрачена, как бы ни была хороша база данных. Кроме того, такие опасности для компьютера, как его перегрев или природные катаклизмы, могут уничтожить ваши данные путем вывода из строя жестких дисков, на которых хранятся эти данные.

Поэтому большинство баз данных предлагают резервное копирование и дублирование. Резервное копирование — это просто создание копии базы данных, позволяющее при каких-либо неурядицах восстановить базу данных из резервной копии и вернуть всю утраченную информацию (или по крайней мере основной объем этой информации).

Дублирование предполагает повторение всей базы данных и, возможно, также запуск этой версии-дубликата. Поэтому вдобавок к наличию основной базы данных и потенциальной резервной копии вы получаете еще копию запущенной базы данных. При использовании дублирования может быть остановлена вся база данных, но все ваши приложения продолжат работу, поскольку они могут переключиться на использование дубликата.

Дублирование является довольно дорогим удовольствием, поскольку для него нужен еще один сервер с еще одной копией запущенного программного обеспечения базы данных. Тем не менее, если ваше приложение подвергается интенсивной эксплуатации, дублирование является надежным способом обеспечения сохранности операций при чрезвычайной ситуации.

Вы постоянно работаете с чем-то подобным на своем компьютере: с системой, хранящей вашу информацию длительное время. Это жесткий диск и файловая система. Все файлы, имеющиеся в вашем компьютере, — это файлы адресов, почтовых сообщений, финансовых документов, уровней ваших игр и других фрагментов информации. И вы можете выключить компьютер и снова его запустить или даже обновить его компоненты и сохранить при этом неприкосновенность данных.

Поэтому файловая система является настоящей разновидностью базы данных. Фактически многие базы данных используют файлы практически так же, как и компьютер, обеспечивая постоянство своей информации. А почему же тогда PHP не хранить информацию просто в файлах? В конце концов у этого языка есть полный набор инструментов для работы с файлами, включая их создание, запись и чтение. Разве этого недостаточно? Оказывается, недостаточно. Чтобы понять почему, читайте дальше.

ПРИМЕЧАНИЕ

Как использовать PHP для работы с файлами, будет рассказано в главе 4.

База данных — это в первую очередь структура

Если подумать, то файловая система компьютера слишком неуклюжа. Вы пытались когда-нибудь запомнить, когда в последний раз посылали кому-нибудь сообщение по электронной почте? Вы не можете зайти в профиль этого человека в адресной книге, если она не подключена к вашей программе электронной почты. Да и ваша программа электронной почты вряд ли поможет, если вы не помните точный электронный адрес этого человека.

Затем, даже если вы нашли нужное сообщение электронной почты, вам могут понадобиться ссылки на упомянутые там документы. И где их искать? Где-нибудь в другой папке, возможно, в какой-нибудь структуре, о которой вы уже давным-давно позабыли.

Именно поэтому компьютер имеет массу разнообразных поисковых возможностей. На Mac OS X можно воспользоваться Spotlight (рис. 3.1) или чем-нибудь вроде QuickSilver (http://quicksilver.en.softonic.com/mac). Spotlight на Mac OS X может связать файлы в различных местах по их именам, по папкам, в которых они находятся, или по их содержимому. Иными словами, Spotlight ведет поиск с целью определения связей между различными файлами и папками.

Пользователи Windows могут загрузить Desktop Search от Google (www.google.com/quicksearchbox) (рис. 3.2). Google Desktop Search работает как на Windows, так и на Macintosh. Программа пытается индексировать и связывать файлы как на вашей машине, так и в хранилищах Google Documents и Gmail. Практически она выстраивает свою собственную базу данных для создания и запоминания всех этих связей. Она ищет все появления конкретного слова или темы на всей вашей системе.

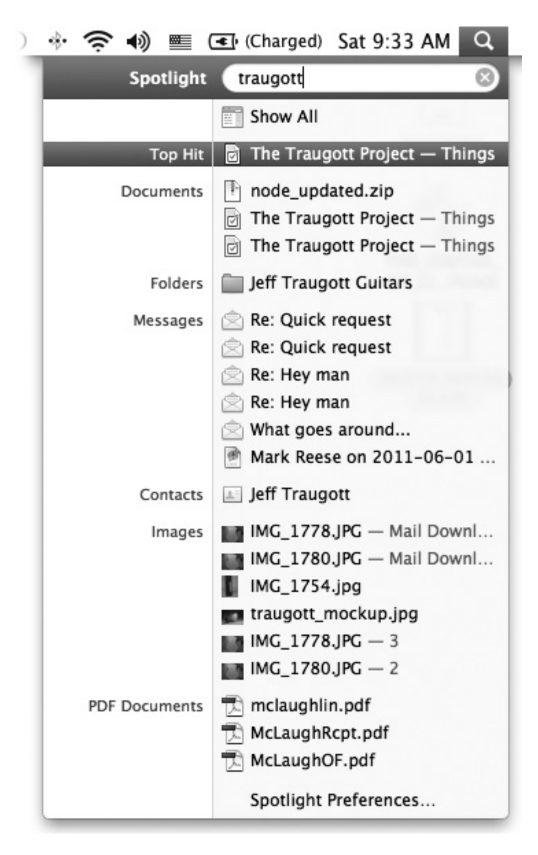


Рис. 3.1. Поиск с помощью Spotlight

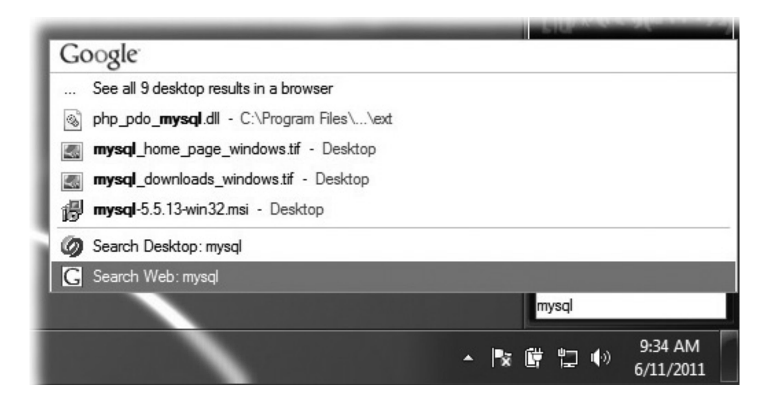


Рис. 3.2. Поиск с использованием Google Desktop Search

Фактически эти поисковые программы пытаются сделать работу, присущую базам данных: найти и структурировать информацию. Но если вы когда-либо пытались создавать такие виды подключений на компьютере с использованием

Spotlight или Google Search или просто вручную, то знаете, что при этом все равно возникают какие-то трудности и непоследовательности. Поэтому нужен более подходящий способ объединения двух, трех или десяти информационных фрагментов.

(Хорошие) базы данных являются реляционными

То, чего недостает файловой системе и жесткому диску, является главным достоинством базы данных. Речь идет о связях между различными фрагментами информации. Например, ваш знакомый имеет несколько адресов электронной почты, телефонных номеров и простых почтовых адресов. Такого рода связи обрабатываются вашей программой адресной книги.

Но хорошая база данных идет еще дальше. Сообщение электронной почты имеет отношение к адресу его отправителя, а этот адрес имеет обратное отношение к имени человека, его телефонным номерам и другой контактной информации. И, разумеется, карта, на которой помечены улицы, связывает их с адресами контактов. И имя создателя в описателе файла связано с этим человеком и его электронным адресом, его телефонным номером и т. д.

Благодаря всевозможным способам эти связи являются настоящей гигантской сетью подключений. И хорошая база данных и создает все эти связи, и управляет ими. Фактически MySQL и другие, наиболее часто встречающиеся базы данных, настолько полагаются на связи (или отношения, по-английски — relation), что это стало частью имени этой категории баз данных: *реляционные базы данных*. (Дополнительная информация о категории базы данных, приспособленной под PHP, дана во врезке «Дополнительные факты. Объекты и связи в базах данных» следующего раздела.)

Следовательно, все это означает, что помимо сообщения базе данных о том, какую информацию нужно сохранить для себя и своих программ, вы также сообщаете, как эта информация подключается к другим информационным частям. Вам приходится не только пользоваться этой сетью подключений, но также сообщать базе данных о том, как именно должна быть сконструирована эта сеть. Это очень мощная технология, поэтому для работы с реляционными базами данных вам нужно выучить абсолютно новый язык.

Установка MySQL

Но перед тем как разбираться с новым языком, нужно установить базу данных. Из названия книги следует, что работать предстоит с MySQL, одной из самых распространенных баз данных, применяемых в веб-приложениях. Ее частое применение обусловлено простотой получения, легкостью установки и простотой использования.

ПРИМЕЧАНИЕ

Как и многое другое в жизни, простота применения сопровождается некоторыми компромиссами. Существуют довольно дорогие и сложные в использовании базы данных, вроде Oracle. Но такие базы данных обычно предлагают свойства, которых нет в таких программах, как MySQL: более качественные инструменты для обслуживания и полный набор возможностей профессиональной поддержки, которые выходят за пределы возможностей, получаемых от MySQL.

Тем не менее не стоит переживать. Практически любая отдельная команда, технология и инструмент, изученный для работы с MySQL, будет работать с любой реляционной базой данных. Поэтому, если вы окажетесь в ситуации, где будет использоваться Oracle (или программный продукт IBM, PostgreSQL или что-нибудь совершенно другое), проблем с настройкой вашего кода PHP под работу с базой данных, отличной от MySQL, не будет.

ДОПОЛНИТЕЛЬНЫЕ ФАКТЫ

Объекты и связи в базах данных

Десятилетиями реляционные базы данных были де-факто стандартом для профессиональных приложений, независимо от того, где они работали: в Интернете или во внутренней сети компании. Эти базы данных часто назывались RDBMS (Relational Database Management Systems, системы управления реляционными базами данных), и большинство данных в этих базах вписывались в предложенную RDBMS-модель. Но дело не только в этом. RDBMS, по сравнению с базами данных другого типа, были более стабильными и профессионально отработанными системами.

Но в последнее время у них появились конкуренты. В большинстве своем это объектно-ориентированные системы управления базами данных (Object-Oriented Database Management Systems, OODBMS). Хотя OODBMS появились еще в 70-х годах прошлого века, популярность они набрали в последние 10 лет.

RDBMS хранят информацию в таблицах, строках и столбцах. Например, у вас может быть таблица пользователей со столбцами для имен, фамилий и адресов электронной почты. Все, что хранится в RDBMS, требует некой разновидности отображения, поэтому информация в PHP-сценарии должна быть отображена на конкретные таблицы и столбцы. Можно, к примеру, сказать, что информация в `$_REQUEST['first_name']` должна быть сохранена в таблице `Users`, а затем в столбце `first_name`. Это двухшаговое отображение не создает больших проблем, но является дополнительным шагом в работе с реляционными базами данных.

В OODBMS вы создаете объект в коде, который замещает таблицу со столбцами. Поэтому вы можете создать новый объект `User` и присвоить его имени значение, хранящееся в `$_REQUEST['first_name']`. Затем, когда понадобится сохранить эту информацию о пользователе, вы просто отдаете OODBMS ваш объект `User` целиком. Иными словами, база данных определяет, что делать с объектом, и ей не нужно сообщать, куда помещать конкретные блоки информации.

Разумеется, при работе с OODBMS это означает необходимость иметь множество объектов в коде, поэтому и при работе с RDBMS, и при работе с OODBMS придется в конечном итоге создавать некий код. Модель RDBMS, используемая MySQL, более распространена в веб-приложениях, чем OODBMS, поэтому именно на ней и нужно остановиться при изучении.

MySQL для Windows

Установка MySQL для Windows не представляет особого труда. Нужно только знать одну вещь: в какой версии Windows работает компьютер: 32- или 64-разрядной. Это можно определить, нажав кнопку Пуск, щелкнув правой кнопкой мыши на пункте Компьютер, а затем выбрав в контекстном меню пункт Свойства. В результате откроется окно (рис. 3.3).

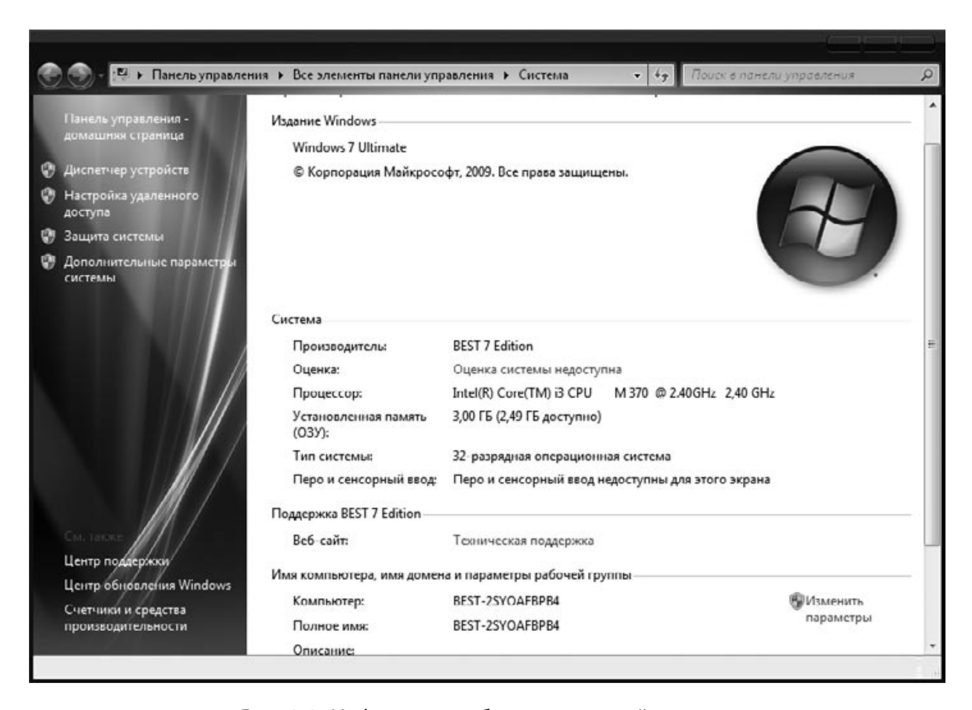


Рис. 3.3. Информация об установленной системе

ПРИМЕЧАНИЕ

Если у вас Macintosh, перейдите к следующему подразделу.

Найдите строку с надписью Тип системы. В ней должно быть написано либо *32-разрядная операционная система*, либо *64-разрядная операционная система*. Запомните, что там написано, поскольку это пригодится вам буквально через минуту. Например, показанная на рис. 3.3 машина является 32-разрядной системой, на которой запущена Windows 7 Professional. То, что вы видите в этом окне, определяется той версией Windows, которую вы установили, а также возможностями вашего компьютера. Запуск MySQL без всяких проблем возможен как на 32-, так и на 64-разрядных системах.

Теперь зайдите через свой веб-браузер на адрес mysql.com (рис. 3.4). На этой странице много вводной информации о MySQL, которую можно либо прочитать,

либо пропустить. Несколько лет назад база данных MySQL перешла из разряда проектов с открытым источником в разряд проектов, поддерживаемых компанией. База данных по-прежнему имеет свободное распространение, но теперь для MySQL появилось намного больше профессиональных систем поддержки. Большинство из них предлагаются на веб-сайте mysql.com: профессиональная поддержка и документация.

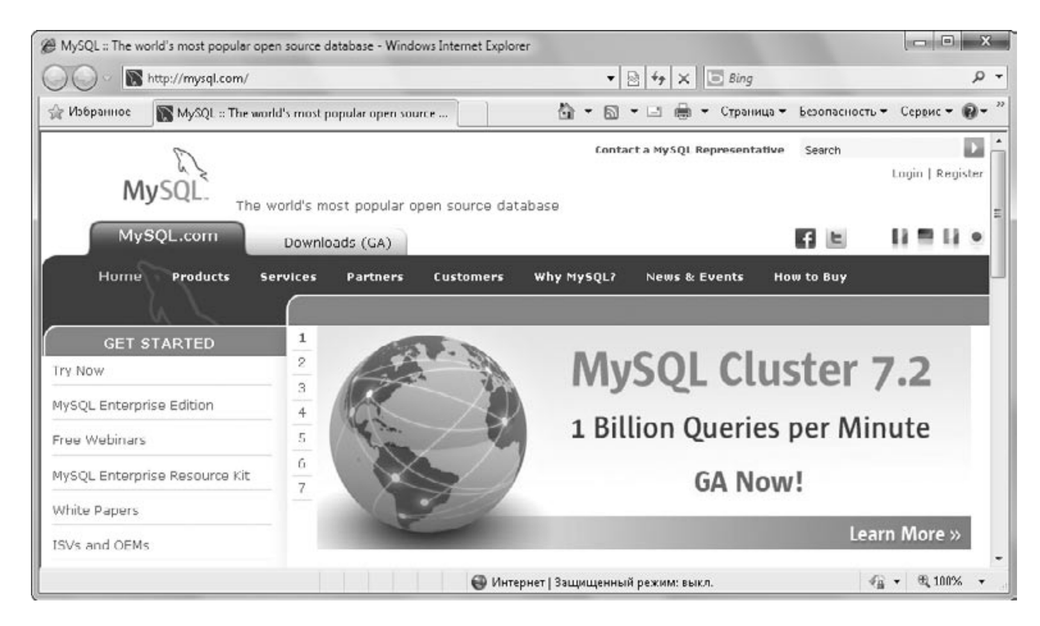


Рис. 3.4. Сайт, посвященный MySQL

Щелкните на большой вкладке Downloads (GA) (Скачать). Откроется сраница со сведениями о нескольких разных версиях MySQL. Вам нужна только одна из них — MySQL Community Server, поэтому щелкните на ссылке Download (Скачать) под этим вариантом. Страница загрузки определит, что вы работаете в Windows, и предоставит несколько вариантов установки (рис. 3.5). Как правило, наилучшим выбором будет MSI Installer, соответствующий вашей системе. У варианта Zip archive не такая удобная упаковка. Итак, вам нужна версия, предлагающая MSI-установщик и соответствующая типу вашей системы: 32-bit или 64-bit. После выбора нужной версии вас попросят зарегистрироваться на веб-сайте MySQL. Этот этап можно пропустить. Если вы переживаете, что в один прекрасный день команда MySQL воспользуется вашим физическим адресом в грязных целях, можно перейти непосредственно к серверам загрузки.

В конечном итоге вы получите перечень серверов, с которых можно загрузить MySQL. Нужно просто выбрать один, самый близкий географически (рис. 3.6), выбрать место для загрузки на вашем компьютере и существенно чем-нибудь подкрепиться, поскольку предстоит еще много работы.

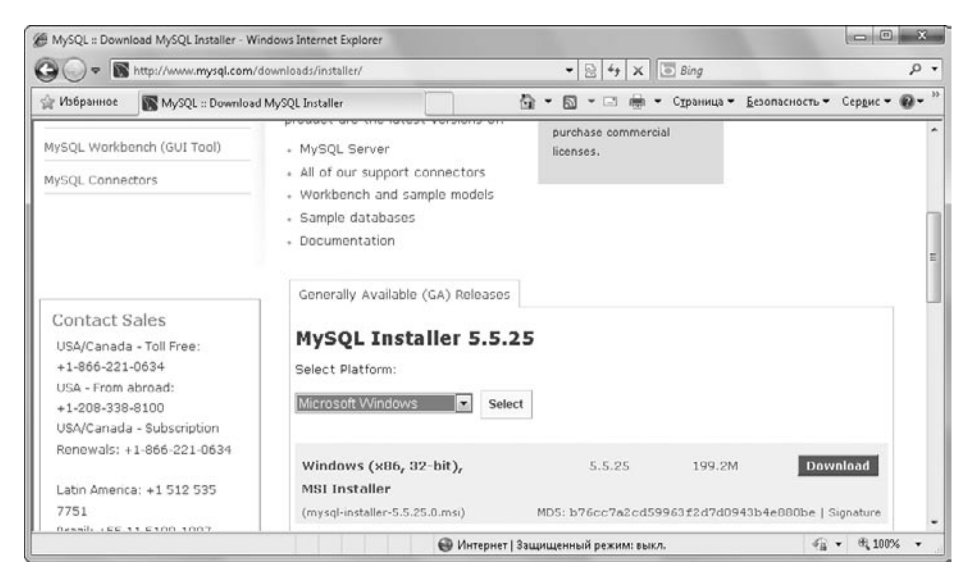


Рис. 3.5. Страница с вариантами установщиков MySQL

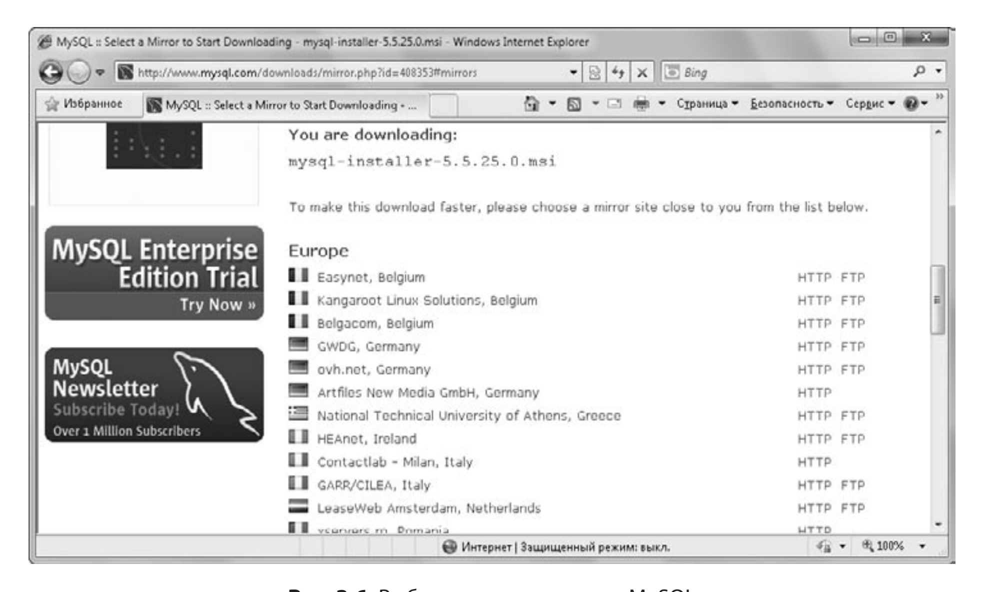


Рис. 3.6. Выбор сервера загрузки MySQL

После завершения загрузки у вас должен появиться файл с именем, похожим на `mysql-5.5.13-win32.exe`. Чтобы запустить установщик, нужно дважды щелкнуть на этом файле. Мастер установки попросит вас принять лицензионное соглашение, а затем даст возможность выбрать тип установки. Нужно выбрать Typical (Обычная).

В процессе установки вспомогательного набора программ нужно будет несколько раз щелкнуть, чтобы инсталляции продолжилась. После завершения установки

появится возможность запустить мастер MySQL Server Instance Configuration Wizard. MySQL сама по себе заслуживает написания толстой книги. Чтобы добиться от нее более быстрой и менее напрягающей вашу систему работы, можно настроить буквально сотни параметров. Но для наших целей все эти сложности ни к чему: нам нужна всего лишь локальная база данных для хранения информации. Тем не менее рассмотрим несколько параметров, чтобы добиться слаженной работы MySQL и вашего компьютера.

В окне мастера конфигурации нужно выбрать стандартную конфигурацию (Standard Configuration) (рис. 3.7). Затем следует разрешить MySQL установиться в качестве службы Windows, что будет означать возможность непосредственного доступа Windows к MySQL и управления этой базой данных. Необходимо также оставить установленным флажок Launch the MySQL Server automatically (Автоматически запускать MySQL-сервер), чтобы MySQL запускалась при каждом запуске компьютера. Нужно также установить флажок для добавления MySQL-каталога bin к переменной поиска пути Windows (Include Bin Directory in Windows PATH) (рис. 3.8). Эта настройка гарантирует запуск программ MySQL из окна командной строки. Кроме того, MySQL поставляется вместе с рядом инструментов, которые позволяют запускать ее базы данных, останавливать их и работать с ними. Все они станут легкодоступны, если добавить каталог bin при установке MySQL к переменной поиска пути (PATH).

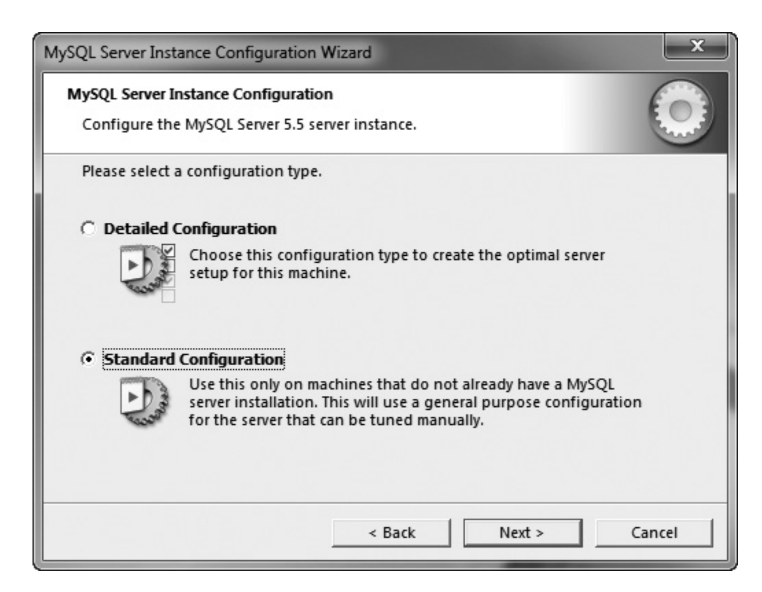


Рис. 3.7. Выбор стандартной конфигурации

Затем нужно ввести корневой пароль (Root Password), являющийся, по сути, главным паролем. Если бы это была реальная база данных, запущенная на сервере сайта www.amazon.com или www.zappos.com, то там не обошлось бы без применения запутанного 22-символьного пароля, который не мог бы взломать самый мощный



Рис. 3.8. Первоначальная настройка MySQL

компьютер. Но сейчас вы запускаете MySQL на своей машине, поэтому в качестве пароля подойдет что-нибудь менее устрашающее, например `myqsl_root`. И наконец, MySQL готова выполнить заданные вами настройки. Щелкните на кнопке Execute (Выполнить), чтобы запустить выполнение.

ПРИМЕЧАНИЕ

Вы, наверное, уже задумывались о том, почему большинство встречающихся вам программистов отличаются нетерпеливостью, легкой нервозностью и неумеренным потреблением кофе. Это из-за того, что им приходится слишком долго ждать, когда дело доходит до установки программного обеспечения, и еще больше ждать, когда дело дойдет до запуска их программ и получения возможности убедиться, что их поведение носит вполне ожидаемый характер.

В конечном итоге окно мастера закроется и база данных MySQL будет установлена. Если щелкнуть на кнопке Пуск, вы увидите, что стала доступна новая программа MySQL Command Line Client (рис. 3.9). Если MySQL Command Line Client не виден в меню Пуск, можно просто открыть окно командной строки и набрать `mysql`. Эта команда открывает клиента командной строки, если, конечно, вы позаботились о добавлении MySQL-каталога bin к Windows-переменной `PATH` в процессе установки MySQL (см. рис. 3.8).

Откройте MySQL Command Line Client и введите свой самый секретный пароль. Программа командной строки всегда начинает свою работу с запроса вашего пароля. Парольная защита играет важную роль для этой программы, поскольку с ее помощью можно делать все: от создания и удаления структур до работы с данными MySQL. Это похоже на прямую линию доступа к MySQL, а это именно то, что вам нужно для тестирования вашего кода PHP, который вы начнете создавать в данной главе.



Рис. 3.9. MySQL Command Line Client в меню Пуск

Итак, в результате вы должны получить нечто похожее на то, что показано на рис. 3.10.

MySQL 5.5 Command Line Client

```
Enter password: ***********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.5.13 MySQL Community Server (GPL)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Рис. 3.10. Окно MySQL Command Line Client

Если вы можете войти в MySQL, значит, получаете запущенную базу данных и уже готовы приступить к работе с этой базой данных и помещать в нее информацию.

MySQL для Mac OS X

Процесс установки MySQL в Mac OS X похож на инсталляцию в Windows. Зайдите на веб-сайт www.mysql.com и выберите вкладку Downloads (GA) (Скачать) в верхней части страницы. Затем щелкните на ссылке MySQL Community Server. Сайт определит, что вы работаете под управлением Mac OS X, и загрузит список доступных версий.

СОВЕТ

Если вы работаете под Windows, вернитесь к разделу «MySQL под Windows».

ПРИМЕЧАНИЕ

Как и в случае с версиями для работы под Windows, MySQL для Macintosh предоставит вам широкое поле выбора. Разработчики, использущие MySQL, склоняются к варианту Compressed TAR Archive, поскольку он содержит действующий код MySQL. Если вы не планируете работать с действующим кодом MySQL, то вам не нужна эта версия.

Прокрутите страницу и найдите ссылки DMG. Это версия MySQL с упрощенной установкой, предоставляющая красивый интерфейс инсталляции. Вам следует определить, на какой системе вы работаете: 32- или 64-разрядной. Для этого сделайте следующее.

Выполните команду ▸ About This Mac (Об этом Mac), а затем щелкните на кнопке More Info (Дополнительная информация), чтобы открылось окно, показанное на рис. 3.11. Найдите строку Processor Name (Имя процессора).

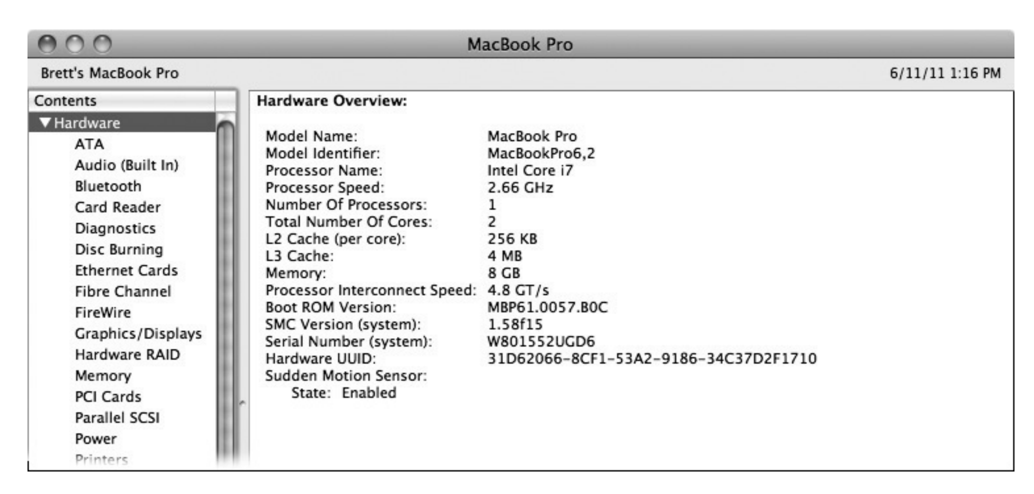


Рис. 3.11. Окно информации о системе

Разрядность системы Macintosh (32 или 64 разряда) можно определить по типу установленного процессора. По этой причине вам нужно знать, какие процессоры 32-, а какие 64-разрядные. К счастью, вам не нужно возиться с большим количеством вариантов. Каждый процессор для компьютеров Macintosh может быть или только 32-разрядным, или 64-разрядным. В табл. 3.1 показаны возможные варианты.

Таблица 3.1. Выриванты процессоров для компьютеров Macintosh

Имя процессора	Разрядность
Intel Core Solo	32
Intel Core Duo	32
Intel Core 2 Duo	64
Intel Quad-Core Xeon	64
Dual-Core Intel Xeon	64
Quad-Core Intel Xeon	64
Core i3	64
Core i5	64
Core i7	64

ПРИМЕЧАНИЕ

Apple постоянно обновляет варианты оборудования Macintosh. Если вы не можете найти название своего процессора в табл. 3.1, зайдите на веб-сайт http://support.apple.com/kb/HT3696. На нем приведен актуальный список процессоров с указанием их разрядности.

Выберите DMG-загрузку, соответствующую вашему процессору. Затем вы можете зарегистрироваться на сайте (или пропустить регистрацию), выбрать сайт, с которого будет осуществляться загрузка, и запустить загрузку.

После загрузки DMG-установщик будет открыт автоматически. Вы увидите несколько файлов (рис. 3.12). Большинство DMG-установщиков имеют один файл и, если повезет, парочку скудных инструкций. Но в MySQL в дополнение к основной установке вы получаете панель предпочтений (которую инсталлируете через несколько минут), программу для управления автоматической установкой и толковый файл `ReadMe.txt`.

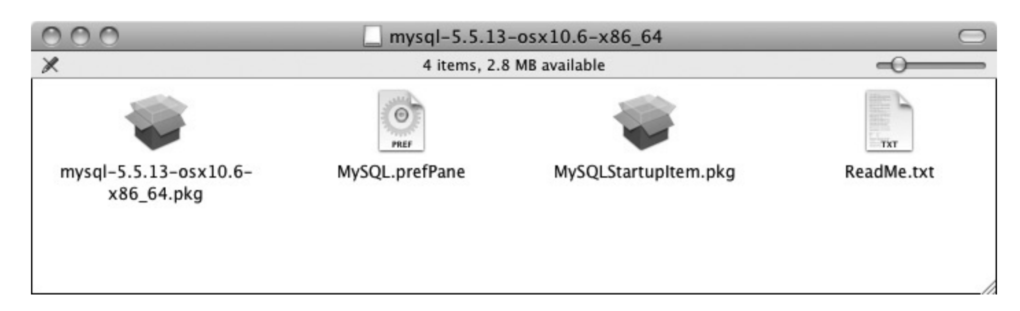


Рис. 3.12. Файлы DMG-установщика MySQL

Выберите главный файл, название которого должно быть похоже на `mysql-5.5.13-osx10.6-x86_64.pkg`. Дважды щелкните на нем для начала установки. Вам нужно будет согласиться с условиями лицензии и выбрать место установки. Затем необходимо набрать пароль администратора для вашей машины, чтобы запустить установку.

СОВЕТ

Если вы работаете на своей машине, этот пароль, скорее всего, совпадает с тем, с которым вы обычно входите в систему. Компьютер Macintosh, которым пользуется всего один человек, объявляет этого пользователя администратором. В противном случае купите тортик, чтобы задобрить владельца компьютера, на котором вы хотите поставить PHP и MySQL.

MySQL устанавливается не только как программа, но и как системный уровень. Она должна иметь возможность не только записывать ваши файлы, но позволять обращение к ней из командной строки Macintosh, пользоваться системными ресурсами и делать многое другое. Сама инсталляция не займет много времени (рис. 3.13). Но особо радоваться не стоит, нужно будет выполнить еще несколько шагов. Вернитесь к установщику DMG (см. рис. 3.12). Если он еще не открыт, дважды щелкните на его значке, чтобы снова открыть окно.



Рис. 3.13. Установка MySQL

После установки панель предпочтений (Preferences Pane) откроется автоматически (рис. 3.14). Панель предпочтений — удобное средство, позволяющее запускать

и останавливать базу данных. При возникновении проблем она предоставляет вам удобное место для определения их причин.

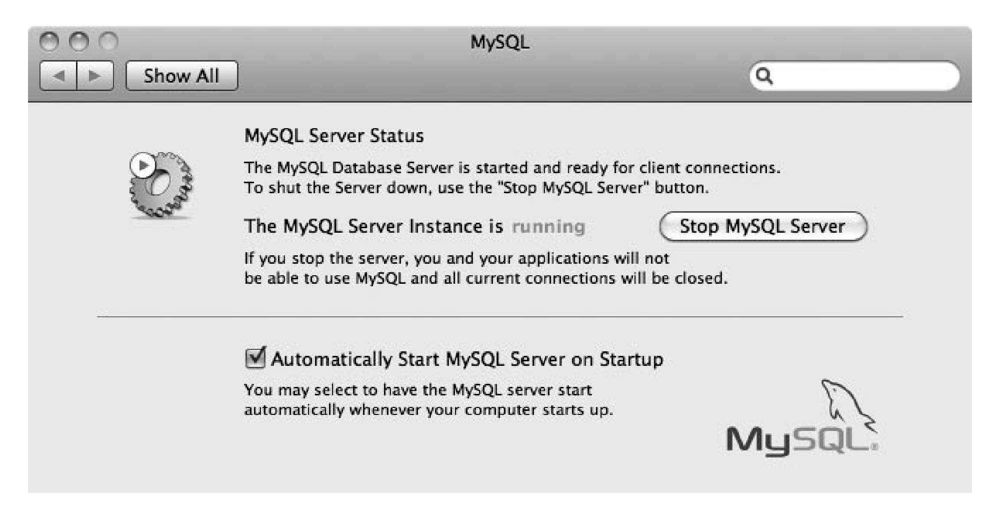


Рис. 3.14. Панель предпочтений MySQL в Mac OS X

Установите флажок автоматического запуска MySQL (Automatically Start MySQL Server on Startup), для чего вам придется ввести свой пароль еще раз. И наконец, запустите MySQL, чтобы убедиться в ее работоспособности.

Теперь, когда установка завершена, запустите базу данных на своей Macintosh-машине. Откройте новое окно терминала (для чего выполните команду Applications ▸ Utilities ▸ Terminal (Приложения ▸ Утилиты ▸ Терминал); а если вы еще не перетащили значок Terminal на панель быстрого доступа, то сделайте это сейчас). В окне Terminal наберите следующую команду:

```
$ /usr/local/mysql/bin/mysql
```

К сожалению, эта команда немного длинновата. Причина в том, что установка не настроила путь поиска каталога на упрощенный вызов инструментов и программ MySQL. (Вероятнее всего, основной объем работы MySQL будет осуществляться на вашем веб-сервере, поэтому это не играет особой роли. Но можно сделать так, чтобы вы могли просто набрать только часть этой команды — mysql; см. следующую врезку.)

Только что набранная вами команда открывает инструмент командной строки MySQL (рис. 3.15). Для компьютеров Macintosh есть графические инструменты, позволяющие работать с базой данных, которые вам обязательно захочется проверить в деле. Но чтобы заглянуть в самый корень сложной проблемы или понять, как нужно работать с MySQL в PHP, не остается ничего другого, как изучить команды, которые можно использовать в командной строке MySQL, предназначенные для использования самой базы данных.

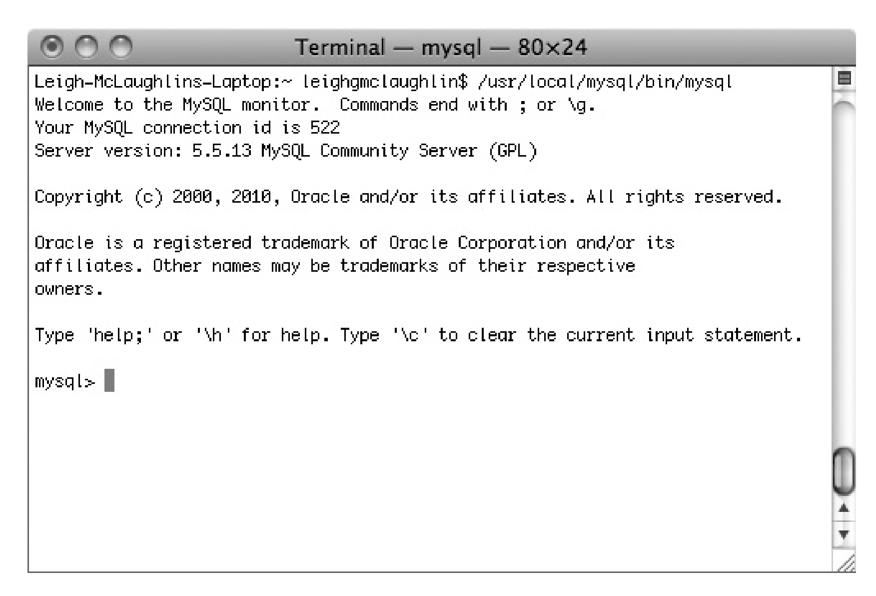


Рис. 3.15. Инструмент командной строки MySQL для Macintosh

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Обновите путь поиска, включив в каталог программ MySQL

Немного разочаровывает то, что несмотря на наличие в MySQL для Macintosh такого удобного средства, как Preferences Pane, вы все равно не можете просто набрать `mysql` в окне Terminal. Но если вы не прочь немного поработать, эту проблему можно устранить своими силами.

Секрет возможности и невозможности запуска всех этих программ в окне Terminal кроется в переменной `PATH`. Это специальная переменная (подобная переменным, которые вы создавали в PHP), сообщающая компьютеру, где нужно вести поиск, когда вы ввели команду. Поэтому когда вы набираете `mysql`, если переменная `PATH` включает путь `/usr/local/mysql/bin`, ваш компьютер ведет поиск в этом каталоге, видит программу под названием `mysql` и запускает ее. Превосходно!

А что делать, когда `PATH` не включает в себя нужный каталог? Вы можете обновить `PATH`, но для этого нужно отредактировать файл, которые обычно скрыт от просмотра. Вернитесь в окно Terminal и введите две команды:

```
$ defaults write com.apple.finder AppleShowAllFiles TRUE
$ killall Finder
```

Первая строка предписывает программе Finder, которая показывает каталоги на Macintosh, отображать скрытые файлы, подобные тому, который нужно отредактировать. Вторая строка перезапускает Finder, вводя тем самым эти изменения в действие. Теперь откройте окно Finder и перейдите в ваш личный каталог. Окно каталога будет показано в немного непривычном виде (рис. 3.16). Там будет масса файлов, выделенных серым цветом, кажущихся блеклыми и почти невидимыми.

Это те самые файлы, которые обычно скрыты от просмотра. Большинство программ, обновляющих систему и работающих с ней, создают скрытые файлы, имена которых начинаются с точки. Так, git, система управления версиями, создает .gitconfig, а DropBox, популярная система совместного использования файлов, создает .dropbox.

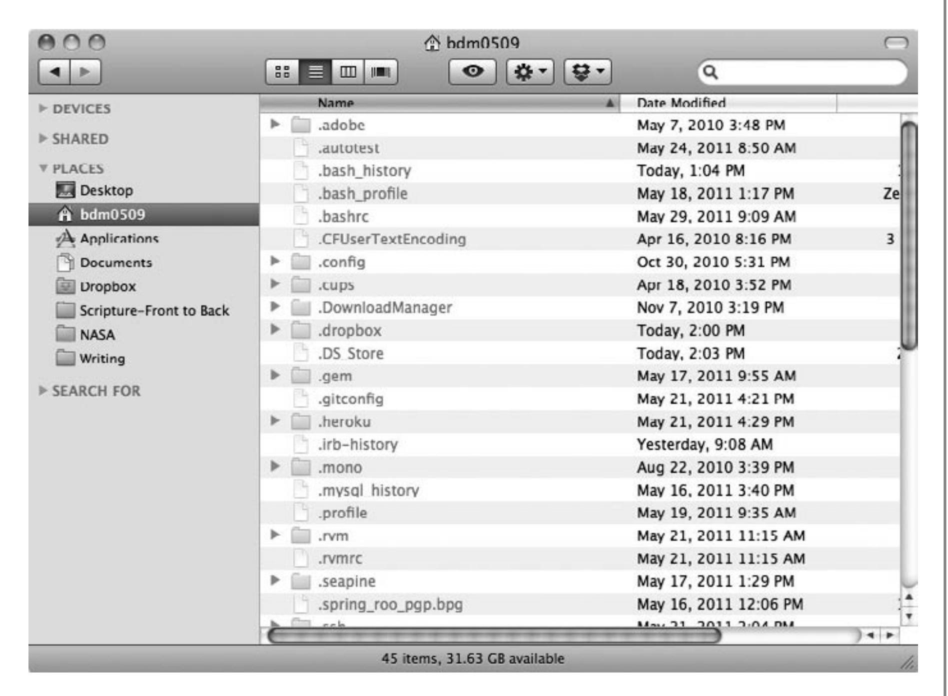


Рис. 3.16. Окно личного каталога

Прокрутите окно, пока не найдется файл по имени .profile. Откройте его в текстовом редакторе, например в TextEdit, имеющемся в Mac OS X. Если вам еще не приходилось работать с PATH, у вас может вообще не быть этого файла, но в этом так же нет ничего страшного. В таком случае просто откройте в TextEdit новый файл.

В этот файл нужно добавить две строки:

```
MYSQL_HOME=/usr/local/mysql
export PATH=$MYSQL_HOME/bin:$PATH
```

Если вы создаете новый файл, эти строки должны стоять первыми. Если у вас уже есть файл .profile, добавьте эти строки в самую нижнюю часть содержимого файла.

В первой строке создается новая переменная по имени MYSQL_HOME, ей присваивается путь к тому месту, куда установлена MySQL. Таким образом, если вы когда-либо измените место установки MySQL, нужно будет просто обновить эту переменную, точно так же, как вы обновляли переменную $facebook_url в PHP-сценарии

(см. пункт «Замена символов в тексте с помощью функции str_replace()» подраздела «Обрезка и замена текста» раздела «Изменение текста» главы 2). Затем во второй строке значение переменной PATH устанавливается на текущее значение этой переменной, но с добавлением в начало пути каталога bin, подчиненного каталогу MYSQL_HOME. Ключевое слово export ставит задачу Mac OS X обеспечить доступность этой обновленной переменной PATH всем программам на вашей машине.

В завершение нужно сохранить ваш файл. Если вы создавали новый файл, ему нужно дать правильное имя, начинающееся с точки (.), и убедиться в том, что он не получил никакого расширения. (Если файл будет случайно сохранен с расширением, просто удалите расширение в окне Finder.)

Когда все будет сделано, в вашем личном каталоге появится файл под названием .profile. Он тоже будет выделен серым цветом, поскольку имеет статус скрытого. Теперь можно открыть новое окно Terminal и набрать mysql. Сразу же должна открыться программа командной строки MySQL.

И наконец, пока вы еще не утратили свои вновь приобретенные навыки мастера редактирования параметров системы, снова настройте Finder на скрытие всех соответствующих файлов:

```
$ defaults write com.apple.finder AppleShowAllFiles TRUE
$ killall Finder
```

Если позже вам понадобится получить к ним доступ, вы всегда сможете включить режим отображения скрытых файлов.

Итак, если вы видите нечто подобное изображенному на рис. 3.15, значит, у вас есть запущенная установка MySQL и вы готовы к работе с базой данных.

Запуск вашего первого SQL-запроса

Убедитесь в том, что MySQL у вас установлена и запущена. При работе в Mac OS X можно проверить Preferences Pane (как было показано ранее на рис. 3.14), а при работе в Windows можно перейти в Панель управления, щелкнуть на пункте Администрирование, а затем выбрать пункт Службы. Прокрутите страницу, пока не увидите службу MySQL, дважды щелкните на этой надписи и убедитесь, что служба имеет состояние Работает (рис. 3.17).

ПРИМЕЧАНИЕ

Если вы уже работаете на веб-сервере, то MySQL, наверное, уже предустановлена и запущена, поэтому можно продолжать работу.

Запустите командную строку MySQL и наберите следующую команду:

```
show databases;
```

ВНИМАНИЕ

Не забудьте в конце строки поставить точку с запятой, иначе вы получите весьма неожиданные результаты. Все команды MySQL заканчиваются точкой с запятой подобно многим командам PHP.

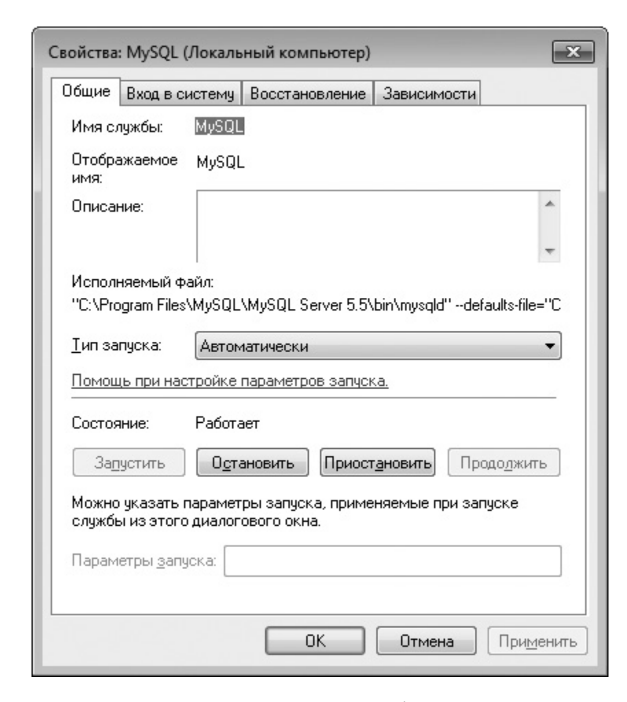


Рис. 3.17. Свойства службы MySQL

Вы получите текстовый ответ от MySQL, который будет иметь примерно следующий вид:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| development        |
| eiat_testbed       |
| mysql              |
| nagios             |
| ops_dashboard      |
| performance_schema |
| test               |
+--------------------+
8 rows in set (0.25 sec)
```

У вас может и не быть так много возвращенных баз данных или же у вас могут быть другие базы данных. Главное, что вам тут нужно увидеть: у MySQL есть заранее созданные базы данных, находящиеся в вашей системе.

А что собой представляла эта команда show? Она выполнила вполне ожидаемое действие: показала вам все, что относится к конкретному ключевому слову, в данном случае — к слову databases. Это просто способ попросить MySQL показать вам все базы данных, установленные на машине.

К тому же вы теперь узнали кое-что действительно важное: MySQL не столько база данных, сколько часть программы, которая может хранить и создавать базы данных. В данном примере команда `show databases;` вернула 8 строк. Значит, в данной системе имеются 8 баз данных, а не одна. Далее вы создадите несколько дополнительных баз данных, которые будут работать в среде MySQL.

А пока сообщите MySQL, что вам нужно работать с базой данных `mysql`, которая имеется в вашей системе, даже если вы только что установили MySQL. Это делается с помощью команды `use`:

```
use mysql;
```

Теперь вы находитесь «внутри» базы данных `mysql`. Иными словами, любые команды в адрес MySQL будут запускаться в отношении базы данных `mysql`.

Вы уже просили MySQL показать все имеющиеся базы данных, а теперь попросите ее показать все таблицы в используемой базе данных:

```
show tables;
```

Вы получите красивый длинный список:

```
mysql> show tables;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| event                     |
| func                      |
| general_log               |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| ndb_binlog_index          |
| plugin                    |
| proc                      |
| procs_priv                |
| proxies_priv              |
| servers                   |
| slow_log                  |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
24 rows in set (0.00 sec)
```

Многие имена этих таблиц выглядят довольно странно, но причина главным образом в том, что это внутренние таблицы MySQL. При создании новых таблиц

или пользователей и после настройки базы данных вся эта информация хранится внутри еще одной базы данных `mysql`.

Чтобы увидеть часть этой информации, нужно выбрать информацию из конкретной таблицы. Можно выбрать всю информацию из таблицы `user`. Для этого в окне командной строки MySQL введите следующую команду:

```
mysql> select * from user;
```

Команда `select` говорит сама за себя: она выбирает информацию из таблицы. Звездочка (*) означает: «Выбери все». Затем `from` сообщает MySQL, откуда взять выбранную информацию: в данном случае из таблицы `user`, принадлежащей вашей базе данных.

Не удивляйтесь, если получите в ответ на эту команду весьма странный поток информации. Он чем-то даже может быть похож на изображение в фильме «Матрица» (рис. 3.18). Когда вы освоитесь с MySQL и PHP, вы научитесь выбирать только нужную вам информацию и очищать подобный беспорядочный ответ.

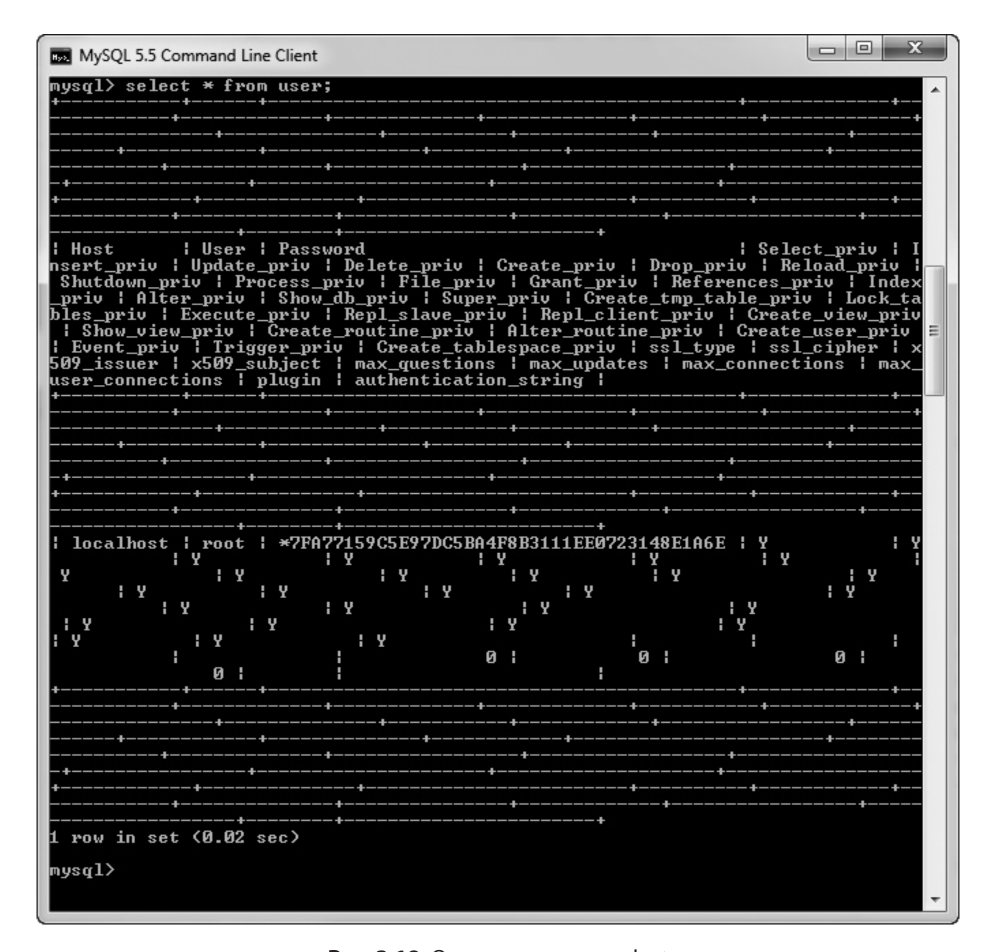


Рис. 3.18. Ответ на команду select

Существуют также способы форматирования ответа от MySQL. Хотя вам не придется сильно волноваться за форматирование, поскольку чаще всего вы будете извлекать информацию из MySQL в сценарий PHP, где форматирование не играет особой роли.

Итак, проблема здесь не в том, что вы набрали в командной строке. Она в том, что вы приказали MySQL выбрать *все* из таблицы user, а в данном случае «все» содержит *слишком много* информации. Фактически в ответе так много информации, что она не может красиво поместиться в окне командной строки, именно поэтому вы получаете все эти странного вида строки.

Чтобы попытаться справиться с этим, вы можете выбрать из таблицы небольшой объем информации. Для этого замените символ * именами нужных столбцов, разделенными запятыми:

```
mysql> select Host, User, Password from user;
```

При этом будут возвращены только три запрошенных столбца:

```
mysql> select Host, User, Password from user;
+------------------------+-------+-----------------------------------------
--+
| Host                   | User  | Password
|
+------------------------+-------+-----------------------------------------
--+
| localhost              | root  | *62425DC34224DAABF6995B46CDCC63D92B03D7E9
|
+------------------------+-------+-----------------------------------------
--+
1 row in set (0.00 sec)
```

В этой таблице показано, что для вашей локальной машины (localhost) имеется один пользователь по имени root. Пароль зашифрован, поэтому ничего полезного вы здесь не увидите, но зато видно, что у MySQL определенно есть вход для вас. Поскольку запрошены были только три столбца, этот ответ читается уже намного лучше и даже имеет некоторый смысл.

Итак, что такое столбец? Это *отдельная категория информации* в вашей таблице. Значит, в таблице, хранящей сведения о пользователях, может быть столбец имени — first_name и фамилии — last_name.

ПРИМЕЧАНИЕ

Пусть вас не смущает обилие новых понятий. В процессе создания PHP-программ вам придется работать с таблицами, столбцами и этими инструкциями MySQL снова и снова. Со временем вы научитесь управляться со всеми этими новыми словами MySQL в мгновение ока.

Теперь, когда вы уже немного познакомились с MySQL, настало время добраться до веб-сервера, приступить к созданию собственных таблиц и столбцов и заполнить эти таблицы и столбцы собственной информацией.

SQL — язык для разговора с базами данных

До сих пор мы занимались тем, что использовали программу по имени MySQL и разговаривали с ней с помощью структурированного языка запросов — SQL (Structured Query Language). И вы уже написали два SQL-запроса:

```
mysql> select * from user;
...
mysql> select Host, User, Password from user;
...
```

Обе эти команды являются SQL-запросами, или просто SQL. Слово «структурированный» появилось в SQL благодаря идее доступа к реляционным базам данных, содержащим множество структур. Вы используете язык, который сам по себе сильно структурирован. Вскоре вы поймете, что SQL очень легко выучить во многом благодаря тому, что он очень предсказуем. Вы можете, например, посмотреть на следующую командную строку и определить, для чего она предназначена:

```
mysql> select User, Password
         from users
        where first_name = 'Dirk'
          and country = 'Germany';
```

Даже если вам никогда не приходилось видеть ключевое слово `where`, все вполне очевидно: запрос возвращает из таблицы `users` только столбцы `User` и `Password` и строки, где поле имени (`first_name`) имеет значение `Dirk`, а поле страны (`country`) — значение `Germany`.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

А зачем устанавливать MySQL на моей машине?

Вы уже видели в первых нескольких главах, посвященных PHP, что большинство создаваемых вами программ предназначено для запуска на веб-сервере. Вы можете вносить ежемесячную плату за хостинг домена на таких серверах, как `jino.ru`, у вас во владении может быть собственный сервер, подключенный к Интернету, или же вы можете развернуть код на серверах своей компании, которые находятся в комнате с кондиционером, где созданы нормальные условия для работы и куда можно попасть только с помощью карточки-ключа. Но во всех этих случаях код оказывается где-нибудь за пределами вашего настольного компьютера или ноутбука.

А если это так (а обычно так все и происходит), зачем утруждать себя установкой PHP и MySQL на своей машине? По правде говоря, вы можете опросить множество PHP-разработчиков, и они признаются, что у них никогда и не было PHP (не говоря уже о MySQL) на их собственных машинах. Они пользуются сессиями telnet и ssh, создавая код на машине, которая находится где-то в недрах Интернета.

Хотя в итоге ваш код редко будет запускаться с вашей собственной машины, существуют вполне резонные причины для установки на ней полного пакета разработчика.

Во-первых, вы не всегда находитесь там, где можно подключиться к Интернету. Вы можете лететь в самолете, ехать на заднем сиденье такси или ночевать где-то в палатке в лесу, имея при себе лишь старый компас в кожаном чехле и MacBook Pro. Во всех этих случаях, если на вашем ноутбуке есть PHP и MySQL, вы можете где угодно программировать, тестировать свой код на реальной базе данных и никогда не сидеть без дела.

Во-вторых, зачастую получается так, что пишется большой объем кода, затем он запускается и обнаруживаются какие-то упущения (иногда весьма серьезные). Код переписывается, проверяется в работе, и все это происходит снова и снова. То же самое получается после того, как вы приступите к обращениям к базе данных. Когда все это делается на сервере, где в конечном итоге и будет размещен ваш код, много времени уходит на передачу данных по сети, расходуются ресурсы данной машины и потенциально происходит добавление данных к базе данных и удаление данных из нее. Гораздо лучше работать на своей собственной машине, а затем на определенном этапе выкладывать весь рабочий код на сервер.

И наконец, в-третьих, устанавливая эти программы с чистого листа, вы узнаете много полезного. Вы лучше разберетесь не только со структурой своей машины, но и с тем, как все эти программы работают. Если у кого-нибудь возникнет конкретная ошибка, вы можете понять, что такая же точно ошибка случалась и у вас, когда не была запущена служба Windows, или у экземпляра MySQL на машине, работающей под управлением Mac OS X, была неправильно настроена таблица прав доступа. Ваш компьютер может помочь вам лучше освоить используемые инструменты, что всегда может пригодиться.

Примеры, приводимые в данной книге, можно запускать и на собственной машине, и на веб-сервере. Если вы работаете на своем компьютере, удостоверьтесь, что вы можете добраться до кода своей машины с помощью веб-браузера. В противном случае выкладывайте код при каждом изменении, чтобы убедиться в его работоспособности.

ВНИМАНИЕ

Споры вокруг произношения аббревиатуры SQL разгораются куда горячее, чем вокруг президентских выборов. Кто-то произносит ее как «сиквел», а кто-то настаивает на произношении «Эс-Кю-Эль», проговаривая все буквы отдельно. Чтобы не выделяться в коллективе сотрудников (трудно быть белой вороной), можете выбрать тот вариант, который прижился в вашей организации.

Вы можете купить книгу по SQL и попробовать запомнить все ключевые слова, но куда лучше будет просто начать создавать собственные таблицы и изучать теорию на практике. Но для этого нужно подключиться к базе данных, чтобы позволить общаться с ней всем вашим PHP-программам.

Вход в базу данных вашего веб-сервера

Освоив азы MySQL, пора приступать к настройкам базы данных, которая используется на вашем веб-сервере. Для входа на веб-сервер вам, вероятно, потребуются такие инструменты, как telnet или ssh.

СОВЕТ

Если вы никогда раньше не пользовались telnet или ssh, введите в поисковике Google любое из имен этих программ, и найдете массу информационных ресурсов. Вам может также потребоваться связаться с владельцами вашего домена и спросить, как лучше получить доступ к вашему серверу. У многих веб-провайдеров сейчас есть графическая версия ssh, которую можно использовать непосредственно на интерактивной панели управления провайдера. Большинство толковых хостинг-провайдеров также имеют подробные интерактивные инструкции, помогающие вам войти в систему и приступить к работе.

После входа в систему откроется доступ к использованию клиента командной строки MySQL — `mysql`. Почти каждый хостинг-провайдер, поддерживающий PHP, поддерживает и MySQL, а это означает, что для начала работы обычно достаточно просто набрать `mysql`.

К сожалению, скорее всего, вы сразу же на выходе получите примерно следующую ошибку:

```
bmclaugh@akila:~$ mysql
ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/
mysql.sock' (2)
```

Сообщения такого рода обычно означают, что MySQL не установлена на вашем сервере или как минимум неправильно сконфигурирована. По-видимому, это сделано намеренно: большинство хостинг-провайдеров хранят свою установку MySQL либо на другой машине, либо, по крайней мере, делают ее доступной только с другого доменного имени, например с http://mysql.kattare.com. Это улучшает защиту, повышает изолированность и безопасность тех баз данных MySQL, которым предоставляется хостинг, что является весьма положительным фактором.

ПРИМЕЧАНИЕ

Если запущенная команда mysql не работает, можно также попробовать воспользоваться командой mysql –hostname=localhost. Некоторые установки MySQL сконфигурированы откликаться только на localhost, а не на то, что называется локальным сокетом. Это повышает защищенность установки MySQL, но в данный момент это не должно вас сильно интересовать. Просто убедитесь в том, что вам удалось тем или иным образом заставить mysql работать.

К счастью, размещение MySQL на другом сервере не создает никаких проблем. Можно запустить команду `mysql` и снабдить ее дополнительной информацией, сообщив, к чему конкретно нужно подключиться. Ключ `--hostname=` позволяет предоставить команде `mysql` имя хоста вашего сервера базы данных MySQL,

а ключ `--user=` дает возможность предоставить `mysql` ваше собственное имя пользователя.

ПРИМЕЧАНИЕ

Ваше имя пользователя для доступа к установке MySQL, предоставляемой провайдером домена, почти наверняка отличается от admin или root. Это имя можно попросить у провайдеров, когда вы будете договариваться о доступе с помощью telnet или ssh. Или, если вы хотите сами попробовать добиться успеха, начните с того имени пользователя и пароля, которые применялись для входа на ваш веб-сервер. Но при этом учтите: у хороших систем управления базами данных будут другие имена и пароли, не такие, как у общающихся с ними веб-серверов.

Объедините все это в командной строке, и у вас получится нечто подобное:

```
bmclaugh@akila:~$ mysql --host=dc2-mysql-02.kattare.com
                        --user=bmclaugh --password
Enter password:
```

Последний ключ `--password` заставляет MySQL запросить у вас пароль. Можно поместить пароль в саму команду, указав ключ вроде `--password=это_не_безопасно`, но тогда у вашего любопытного соседа по кабинету появится возможность войти на ваш MySQL-сервер.

После ввода пароля вы окажетесь в стандартном окне приветствия MySQL, показанном на рис. 3.19.

Теперь вы готовы что-нибудь сделать с помощью нового для вас языка SQL.

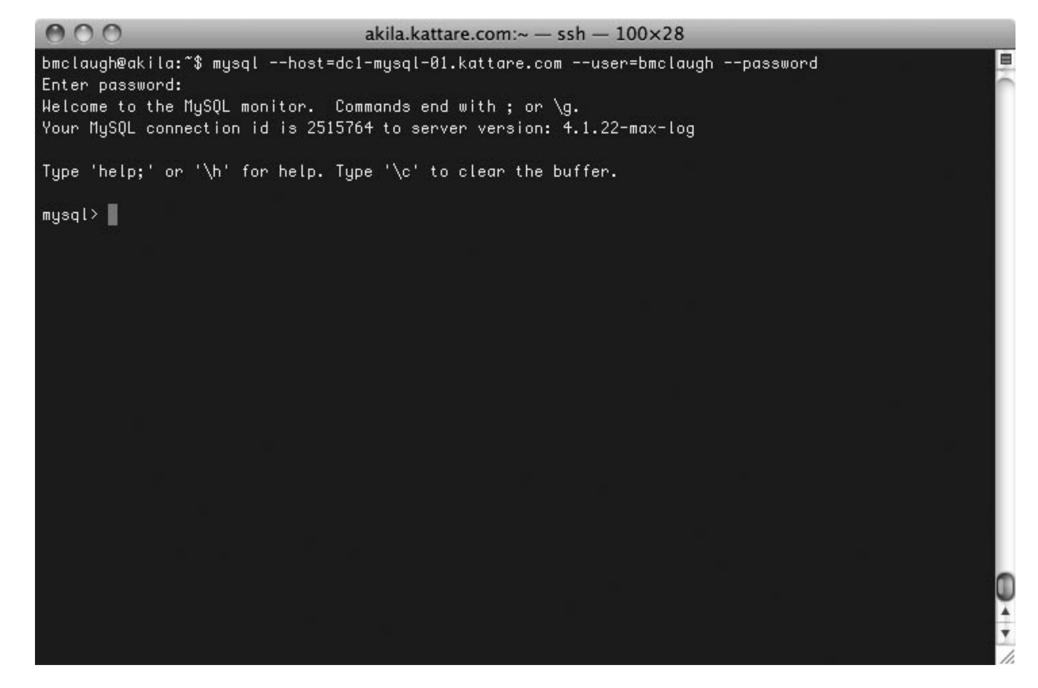


Рис. 3.19. Окно приветствия MySQL

Использование базы данных с помощью команды USE

При использовании большинства баз данных MySQL, предоставляемых хостинг-провайдерами, вы не получите такой же свободы действий, как при использовании MySQL, которая установлена на собственной машине. Наберите, к примеру, ранее использовавшуюся нами SQL-команду:

```
myqsl> show databases;
```

Результат может быть слегка неожиданным. И уж точно он не будет таким, каким вы видели его на своей машине:

```
myqsl> show databases;

+----------+
| Database |
+----------+
| bmclaugh |
+----------+
1 row in set (0.09 sec)
```

Результат имеет усеченный вид, поскольку вы располагаете ограниченными правами на сервере своего хостинг-провайдера. Компания, конечно же, не собирается позволить вам подключиться к своим системным базам данных `mysql` и изучить состав пользователей в ее системной таблице `user`. Скорее всего, вы увидите единственную запись, базу данных с именем, похожим на то, под которым вы вошли в систему. Например, если вы вошли под именем ljuber, можете увидеть базу данных по имени `ljuber`, `db-ljuber` или с каким-нибудь еще похожим названием.

Возможно, эта конкретная база данных уже настроена под вас. По этой причине нужно сообщить MySQL, что эта та самая база данных, с которой вы хотите работать:

```
mysql> use bmclaugh;
Database changed
```

ВНИМАНИЕ

На некоторых системах при входе в MySQL вы автоматически настраиваетесь на использование вашей базы данных. И все же использование приведенной выше команды не помешает, поэтому лучше всегда начинать свою MySQL-сессию с команды use [имя_вашей_базы_данных].

Продолжим знакомиться со средой MySQL. SQL-команды принято писать прописными буквами. То есть фактически те команды, с которыми вы уже сталкивались выше, чаще всего набираются следующим образом:

```
mysql> SELECT * FROM user;
...
mysql> SELECT Host, User, Password FROM user;
...
```

```
mysql> SELECT User, Password
         FROM users
        WHERE first_name = 'Dirk'
          AND country = 'Germany';
```

Благодаря такому формату ключевые слова SELECT, FROM, WHERE и AND легко отличить от имен столбцов и таблиц. Но, как вы уже догадались, MySQL допускает применение ключевых слов как в верхнем, так и в нижнем регистре.

ПРИМЕЧАНИЕ

Хотя мы не обязаны использовать прописные буквы в таких ключевых словах MySQL, как SELECT и WHERE, их применение создает весьма полезный эффект самодокументирования, поскольку выделение в коде ключевых слов помогает быстро улавливать назначение этого кода. Но в действительности многие программисты устают от заглавных букв и набирают буквы в нижнем регистре.

Создание таблиц с помощью инструкции CREATE

При получении доступа к базе данных mysql и ее запуска командой USE у вас уже есть несколько таблиц, в отношении которых можно воспользоваться инструкцией выбора SELECT, например таблица users. Но теперь вы находитесь на сервере базы данных, на котором вы не можете получить доступ к этим таблицам. Поэтому прежде, чем вернуться к отработке приемов применения инструкции SELECT, нужно создать какую-нибудь таблицу.

Как вы уже могли догадаться, это можно сделать с помощью другого простого ключевого слова SQL — CREATE. Итак, вам нужно создать таблицу. Затем вы сможете поместить в нее данные, извлечь эти данные и в целом можете делать с ней что угодно.

Наберите в командной строке MySQL следующую команду:

```
CREATE TABLE users (
```

Не ставьте в конце команды точку с запятой. Затем нажмите клавишу Enter, и увидите нечто необычное:

```
mysql> CREATE TABLE users (
    ->
```

Что же происходит? Как вы уже знаете, команды MySQL завершаются точкой с запятой. Но здесь этот знак не поставлен. Тем самым программе MySQL сообщается: «Я написал команду, но еще не завершил ее». Иными словами, вам не нужно заталкивать слишком длинную программную строку SQL в одну строку своего инструмента, вы можете разбить ее на несколько строк и просто продолжать нажимать клавишу Enter. Поскольку вы не набрали точку с запятой, MySQL не будет

пытаться что-то сделать с вашей командой. И эта небольшая стрелка (->) сообщает о том, что MySQL ожидает от вас продолжения набора.

Поэтому продолжайте набор команды:

```
mysql> CREATE TABLE users (
    -> user_id int,
    -> first_name varchar(20),
    -> last_name varchar(30),
    -> email varchar(50),
    -> facebook_url varchar(100),
    -> twitter_handle varchar(20)
    -> );
```

После этой последней точки с запятой нажмите Enter и получите совсем невыразительный ответ:

```
mysql> CREATE TABLE users (
    -> user_id int,
    -> first_name varchar(20),
    -> last_name varchar(30),
    -> email varchar(50),
    -> facebook_url varchar(100),
    -> twitter_handle varchar(20)
    -> );
Query OK, 0 rows affected (0.18 sec)
```

Последняя строка говорит: «Я сделала то, что ты просил».

Итак, рассмотрим, что происходит в нашей команде `CREATE`.

- `CREATE` сообщает MySQL, что вы хотите создать в базе данных новую структуру.
- `TABLE` сообщает MySQL, какого рода структуру нужно создать. В данном случае нужна таблица.
- `users` является именем создаваемой таблицы.
- Открывающая круглая скобка `(` говорит MySQL о том, что вы собираетесь дать построчное описание создаваемой таблицы.
- В каждой строке имеются имя столбца, например `user_id`, и тип, например `int` или `varchar(20)`.
- После завершения описания таблицы используется закрывающая круглая скобка `)`, оповещающая MySQL об этом завершении, а затем вся эта порция кода завершается точкой с запятой.

Вам еще предстоит изучить многое относительно столбцов разного типа, которые могут у вас иметься, но сейчас нас волнуют только два типа. Первый — `int`, являющийся сокращением слова integer *(целое число)*, то есть 1, 890 и 239 402 относятся к типу `int`, а 1,293 и 3,1456 не относятся.

ПРИМЕЧАНИЕ

В MySQL может применяться не только int, но и integer. Фактически в MySQL эти два слова идентичны.

Назначение следующего типа, varchar, носит не такой очевидный характер. Сокращение varchar означает *строка переменной длины* (variable character), то есть хранение строковых данных (строк) переменной длины. Таким образом, столбец типа varchar(20) может содержать строку длиной от 1 до 20 символов.

Используя все эти новые термины, вы заставили MySQL создать таблицу с несколькими новыми столбцами, один из которых имеет тип int (user_id), а другие — varchar с указанием разной максимальной длины.

А как узнать, что команда CREATE сработала? Для этого можно воспользоваться командой SHOW:

```
mysql> SHOW tables;
+------------------------------------+
| Tables_in_bmclaugh                 |
+------------------------------------+
| users                              |
+------------------------------------+
1 row in set (0.06 sec)
```

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Как исправлять опечатки?

Даже в кодах мастеров PHP и MySQL могут присутствовать опечатки. Фактически, когда программист слишком быстро набирает текст, опечатки становятся в MySQL реальным источником неприятностей. В некоторых случаях MySQL просто выдает сообщение об ошибке и позволяет повторить попытку:

```
mysql> use
ERROR:
USE must be followed by a database name
mysql>
```

В данном случае не произошло ничего страшного. Но в других ситуациях можно допустить ошибку в середине команды или, хуже того, нажать клавишу Enter:

```
mysql> SELECT *,
    ->   FROM
    ->
    ->
```

Здесь в строке SELECT после знака * поставлена лишняя запятая. Но MySQL после каждого нажатия клавиши Enter просто выдает еще одно приглашение на ввод ->.

Проблема в том, что с точки зрения MySQL вы не завершили команду SQL. Поэтому программа не обрабатывает команду, включая и допущенную ошибку, и не дает вам возможности начать все заново.

Попав в подобную ситуацию, лучше будет ввести точку с запятой (;), а затем нажать клавишу Enter. Точка с запятой завершит текущую команду, какой бы неработоспособной она ни была, и предписывает MySQL обработать команду. После этого, скорее всего, будет выведено сообщение об ошибке. По крайней мере вы снова получите управление и сможете внести коррективы.

Итак, вы определенно создали таблицу. Но что в ней находится на самом деле? Чтобы определить это, нужно воспользоваться новой командой: DESCRIBE (описание). Попробуйте воспользоваться этой командой в отношении таблицы users:

```
mysql> DESCRIBE users;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| user_id        | int(11)      | YES  |     | NULL    |       |
| first_name     | varchar(20)  | YES  |     | NULL    |       |
| last_name      | varchar(30)  | YES  |     | NULL    |       |
| email          | varchar(50)  | YES  |     | NULL    |       |
| facebook_url   | varchar(100) | YES  |     | NULL    |       |
| twitter_handle | varchar(20)  | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
6 rows in set (0.04 sec)
```

ПРИМЕЧАНИЕ

Вместо DESCRIBE можно также использовать DESC (или desc). А команда DESCRIBE users; подходит также и в качестве команды SQL.

Теперь вы можете увидеть, что программа MySQL сделала именно то, что ей было приказано: создала таблицу по имени users со всеми указанными столбцами, используя переданные ей описания типов. По поводу другой имеющейся здесь информации можно пока не волноваться.

ЗАМЕТКИ О ПРОЕКТИРОВАНИИ

Размер столбцов особой роли не играет

При создании таблиц многие подолгу задумываются о том, что они хотят хранить в своей базе данных, и почти совсем не думают о таких вещах, какой максимальной длины может достигать поле типа varchar. Поэтому вам будут попадаться таблицы, в которых имеются по 10 или 20 столбцов типа varchar(100), притом что в них содержится совершенно разная по объему и назначению информация.

Но лучше при проектировании таблиц задуматься об этом. Объявляйте нужную длину столбцов, но без превышения. Да, может показаться более безопасным остановиться на заведомо большей длине. Но тогда не получится сделать базу данных соответствующей той информации, которую вы собираетесь в ней хранить.

> Если вы сохраняете имя, то совершенно ни к чему делать максимальную длину имени равной, скажем, длине URL-адреса для Facebook. Вполне нормально будет установить для имени длину в 15 символов (больше просто не понадобится!). И напротив, адрес странички на www.facebook.com в 20 символов едва уместится. Поэтому есть смысл назначать для столбцов разную максимальную длину.
>
> Вопрос длины столбцов имеет отношение только к культуре проектирования и никоим образом не влияет на работу базы данных. База данных использует только то пространство, которое занимают хранящиеся в ней данные, поэтому не нужно испытывать угрызения совести из-за якобы нерационального применения дискового пространства или ухудшения производительности, если все ваши поля типа varchar имеют слишком большую длину. Однако в результате вы получите неаккуратную базу данных, свидетельствующую о том, что вы не слишком задумывались, какая информация будет в ней храниться.
>
> Потратьте время на хорошую конструкцию сейчас, и это воздастся вам сторицей. Задайте для столбцов типа varchar действительную длину, возможно, с небольшим запасом. Но всегда помните о том, какая информация должна поступать в эти столбцы.

Удаление таблиц с помощью команды DROP

Для всего, что позволяет сделать MySQL и SQL, есть способ совершения обратного действия. Вы создали таблицу, но вы также можете ее и удалить. Делается это не с помощью команды delete (удалить), а с помощью команды DROP.

Итак, если будет принято решение, что вам больше не нужна таблица users или что вы хотите еще раз попробовать в деле свою команду CREATE, вы можете избавиться от таблицы users с помощью простой строки SQL-кода:

```
mysql> DROP TABLE users;
Query OK, 0 rows affected (0.10 sec)
```

Все получилось!

```
mysql> SHOW tables;
+-----------------------------------+
| Tables_in_bmclaugh                |
+-----------------------------------+
0 rows in set (0.06 sec)
```

Все очень просто. Но... теперь у вас снова нет ни одной таблицы и не из чего выбирать информацию с помощью команды SELECT. Нужно опять возвращаться к созданию таблиц. Еще раз введите в инструментальное средство MySQL инструкцию CREATE и заново создайте таблицу users.

ПРИМЕЧАНИЕ

Во многих системах для получения в командной строке последней запущенной команды достаточно нажать клавишу со стрелкой вверх. Нажмите клавишу со стрелкой вверх несколько раз, и в командной строке прокрутится история набора команд. Эта прокрутка команд — отличный способ ускорения повторного использования той или иной ранее запущенной команды.

Вставка нескольких строк с помощью команды INSERT

На данный момент вы уже создали, удалили и снова создали таблицу users. Но в ней до сих пор нет ничего, что требовало бы исправления ситуации. Пора уже воспользоваться командой INSERT и вставить в нее немного данных.

Попробуйте набрать в своем инструментарии командной строки следующую команду:

```
mysql> INSERT INTO users
    -> VALUES (1, "Mike", "Greenfield", "mike@greenfieldguitars.com",
    -> "http://www.facebook.com/profile.php?id=699186223",
    -> "@greenfieldguitars");
Query OK, 1 row affected (0.00 sec)
```

Сколько всего нового! Но и в этом SQL можно разобраться, внимательно изучив код. Вы вставляете сведения в таблицу users, а затем по кусочкам выдаете эту информацию.

Вы можете проследить каждое значение и связать его со столбцом в своей таблице. Для этого может понадобиться еще раз получить описание таблицы с помощью команды DESCRIBE:

```
mysql> DESCRIBE users;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| user_id        | int(11)      | YES  |     | NULL    |       |
| first_name     | varchar(20)  | YES  |     | NULL    |       |
| last_name      | varchar(30)  | YES  |     | NULL    |       |
| email          | varchar(50)  | YES  |     | NULL    |       |
| facebook_url   | varchar(100) | YES  |     | NULL    |       |
| twitter_handle | varchar(20)  | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
6 rows in set (0.29 sec)
```

Первое значение, 1, попадает в столбец user_id; второе, "Mike", — в столбец first_name и т. д.

Вот, собственно, и все: вы можете в любое время вставлять в свою таблицу сколько угодно информации. Существует множество вариантов компоновки команды INSERT, большинство из которых вы изучите, когда начнете использовать INSERT в своем PHP-коде.

И в завершение — команда SELECT

И наконец, мы вернулись к возможности использовать нашу старую добрую команду SELECT. Теперь она может показаться чем-то вроде античной истории, поскольку с момента первого использования команды SELECT * FROM user мы уже научились применять DROP, CREATE и INSERT, а также несколько других команд. Итак, попробуем воспользоваться этой командой еще раз:

```
mysql> SELECT * FROM users;
+---------+------------+-----------+---------------------------+--
----------------------------------------------+-------------------
-+
| user_Id | first_name | last_name  | email                     |
facebook_url                                  | twitter_handle
  |
+---------+------------+-----------+---------------------------+--
----------------------------------------------+-------------------
-+
|       1 | Mike       | Greenfield | mike@greenfieldguitars.com |
http://www.facebook.com/profile.php?id=699186223 | @greenfieldguitars
|
+---------+------------+-----------+---------------------------+-
-----------------------------------------------+----------------
---+
1 row in set (0.00 sec)
```

Ничего удивительного не произошло, вы получили назад строку, которую только что вставили. Но как и в предыдущий раз (в подразделе «Запуск вашего первого SQL-запроса» раздела «Установка MySQL»), этот вывод представлен в немного перемешанном виде. Нормальному чтению мешает обилие столбцов.

Для упрощения ситуации извлечем лишь несколько столбцов. Вы уже знаете, как это сделать:

```
mysql> SELECT first_name, last_name, twitter_handle FROM users;
+------------+-----------+--------------------+
| first_name | last_name  | twitter_handle     |
+------------+-----------+--------------------+
| Mike       | Greenfield | @greenfieldguitars |
+------------+-----------+--------------------+
1 row in set (0.00 sec)
```

Этот вывод читать гораздо легче. И поскольку для общения с MySQL используется код PHP, форматирование не будет создавать подобных проблем. PHP не волнует красивое размещение абсолютно всей информации в хорошо читаемых одной-двух строках. Ему вполне подойдет и совсем неаккуратный набор результатов, с которым он справится без всяких проблем.

При желании можете потратить немного времени на вставку еще нескольких строк и на проведение экспериментов с командой SELECT. Если хотите получить настоящую выборку, попробуйте воспользоваться условием WHERE:

```
mysql> SELECT facebook_url
    ->   FROM users
    ->  WHERE first_name = 'Mike';
+---------------------------------------------------+
| facebook_url                                      |
+---------------------------------------------------+
| http://www.facebook.com/profile.php?id=699186223 |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

Если вы еще не совсем поняли назначение ключевого слова WHERE, не стоит волноваться. Просто почувствуйте принцип его работы, испробуйте его в деле и посмотрите, как глубоко можно забраться, располагая уже имеющимися у вас познаниями в SQL.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

SQL или MySQL? Это не одно и то же

Одно дело знать, что означает SQL и как устанавливать MySQL. И совсем другое дело обладать информацией, в чем состоит разница между SQL и MySQL. Поспрашивайте знакомых начинающих программистов, и вы будете удивлены, насколько они вообще не уверены в наличии разницы между языком SQL и программой управления базами данных MySQL.

Фактически SQL — это язык. Он существует отдельно от MySQL или любой другой базы данных, например PostgreSQL или Oracle.

Это означает, что SQL может меняться или обновляться без автоматического внесения изменений в вашу базу данных. Обычно происходит так, что в SQL появляется новое ключевое слово или инструкция, а затем происходит выпуск новых версий всех программ для поддержки этого ключевого слова. Разумеется, SQL не изменяется длительное время, поэтому подобные события случаются нечасто.

MySQL — это программа управления базами данных. Она позволяет создавать базы данных и работать с ними, и эти базы данных допускают применение команд SQL. Существуют и другие программы управления базами данных, которые не допускают применения SQL, и это не делает их в большей или в меньшей степени имеющими отношение к SQL. Фактически MySQL может прекратить использование команд SQL и все равно останется программой управления базами данных (хотя имя утратит свой былой смысл).

Если вы в состоянии различать SQL и MySQL, значит вы готовы к любым неожиданностям. И все благодаря тому, что при работе с PHP вы подключаетесь к базе данных MySQL, но при этом пишете команды и запросы на языке SQL. В результате вы можете поменять базу данных, и практически весь ваш SQL-код будет работать, если только новая база данных допускает использование SQL. Именно в этом и заключается вся прелесть отделения SQL от применяемой вами базы данных, в данном случае — от MySQL. Вы можете заменить эту базу данных, перейдя на PostgreSQL или Oracle, и при этом вам не придется переписывать весь ваш код.

Теперь обратите внимание, что в предыдущем абзаце говорится о том, что *практически весь* ваш код SQL сохранит свою работоспособность. Каждая база данных добавляет собственные поправки в реализацию стандарта SQL. И большинство производителей баз данных добавляют к своим продуктам некоторые особые свойства для придания этим продуктам «дополнительного веса». (Об этом можно прочитать, как о продаже сопутствующего товара.) Поэтому при переходе с одной базы данных на другую могут возникать некоторые проблемы.

Но то, что вы разбираетесь в SQL, поможет и в данном случае. Вы сможете выявить причину любых проблем и быстро найти их решения.

Поэтому изучайте SQL, пользуйтесь MySQL и создавайте такой код, который будет работать на любой базе данных, допускающей применение SQL.

ЧАСТЬ 2

Динамические веб-страницы

Глава 4. Подключение PHP к MySQL
Глава 5. Улучшение поиска с помощью регулярных выражений
Глава 6. Создание динамических веб-страниц

4 Подключение PHP к MySQL

После того как вы увидели, на что способны PHP и MySQL, настало время объединить эти два тяжеловоза. Во многих языках программирования как только речь заходит о базе данных, приходится загружать и устанавливать дополнительный код или инсталлировать небольшие дополнительные модули, предоставляющие программную поддержку для общения с этой базой данных. Но PHP в этом плане не похож на другие языки, он уже поставляется готовым к подключению к MySQL — с того самого момента, когда вы запускаете команду `php`.

Даже если вы начали изучать PHP совсем недавно, вы уже готовы к тому, чтобы использовать базы данных прямо из сценариев. Нужно просто выучить несколько новых команд и научиться справляться с проблемами, возникающими при работе с базами данных. Фактически вы создадите простую форму, позволяющую вводить SQL-команды и запускать их в отношении вашей базы данных MySQL. Программист, работающий на PHP, может обойтись и без окна командной строки `mysql`.

Далее мы рассмотрим взаимодействие PHP и MySQL и напишем еще один сценарий. Он будет получать информацию из всех созданных вами форм, добавлять эти сведения к базе данных, а затем добавлять еще одну форму, позволяющую пользователям вести поиск других пользователей по именам.

Создание простого PHP-сценария, предназначенного для подключения

Неважно, насколько простые или сложные у вас сценарии, если они общаются с базой данных, они начинаются с одних и тех же нескольких действий.

1. Подключение к установленной базе данных MySQL.
2. Использование команды `USE` в отношении нужной базы данных MySQL.
3. Отправка SQL базе данных.
4. Получение результатов.
5. Обработка результатов.

Действия 3, 4 и 5 будут изменяться в зависимости от характера выполняемой работы. Сценарий, создающий таблицы, немного отличается от сценария, который ведет поиск в существующих таблицах.

Но первые два действия — подключение к MySQL и использование нужной базы данных — всегда одни и те же, независимо от предназначения сценария.

Подключение к базе данных MySQL

Сначала нужно сообщить вашему PHP-сценарию, как нужно подключиться к базе данных. Этот процесс, по сути, сообщает PHP, что нужно делать то же самое, что вы выполняли, начиная работу со своим клиентом командной строки MySQL. При подключении к базе данных веб-сервера вы, вероятно, пользуетесь командой, имеющей примерно следующий вид:

```
bmclaugh@akila:~$ mysql --host=dc2-mysql-02.kattare.com
                        --user=bmclaugh --password
```

Чтобы подключиться к базе данных, PHP нужно будет передать точно такую же информацию: имя хоста вашей базы данных, ваше имя пользователя и пароль.

Запустите свой текстовый редактор и создайте новый сценарий, назвав его `connect.php`. Сценарий должен быть как можно проще, потому что вам нужно всего лишь подключиться к своей базе данных, воспользоваться с помощью команды `USE` необходимой базой данных и запустить пробный SQL-запрос, чтобы убедиться, что все работает.

Введите в сценарий следующие строки:

```
<?php
  mysql_connect("хост_вашей_базы_данных",
                "ваше_имя_пользователя", "ваш_пароль")
    or die("<p>Ошибка подключения к базе данных: " .
           mysql_error() . "</p>");

  echo "<p>Вы подключились к MySQL!</p>";
?>
```

ПРИМЕЧАНИЕ

Если база данных запускается на той же машине, на которой находятся PHP и файлы обслуживания сети, хост-именем базы данных будет, как правило, localhost, которое означает «локальная машина».

Вот так все просто! И хотя здесь есть несколько новых для вас команд, вы, наверное, уже знаете почти все, что происходит в этом сценарии.

Сначала в нем используется новая команда: `mysql_connect`. Она просто получает имя хоста базы данных, имя пользователя и пароль и осуществляет подключение. Происходит все то же самое, как и при запуске окна командной строки `mysql` и подключении к удаленной базе данных.

ПРИМЕЧАНИЕ

Не забудьте заменить "хост.вашей.базы_данных", "ваше-имя_пользователя" и "ваш-пароль" на значения для вашей собственной базы данных.

А что можно сказать по поводу той части кода, которая начинается с ключевого слова die? Звучит несколько устрашающе (глагол die кроме всего прочего означает «умирать»). И на самом деле ничего хорошего это слово не означает: die используется, когда что-то в сценарии может не заладиться. О die нужно думать примерно так: «Если мой код не срабатывает, нужно сделать что-нибудь менее раздражающее, чем демонстрация кода ошибки пользователю». В данном случае die выводит сообщение об ошибке, которое не испугает ваших пользователей.

Но перед тем как разбираться с особенностями инструкции die, нужно немного больше узнать о внутренней работе инструкции mysql_connect. Когда запускается mysql_connect, она либо создает, либо заново использует существующее подключение к базе данных. Затем она возвращает это подключение вашей PHP-программе и делает доступными все другие команды перехода от PHP к MySQL, которые вы скоро изучите. Но если mysql_connect не может создать подключение, например если ваша база данных не запущена или вы передали неверное имя хоста или имя пользователя, mysql_connect возвращает совершенно другое значение: false.

Следовательно, в действительности в вашем сценарии происходит примерно следующее:

```
<?php
  // Это не рабочий код, а просто пример
  if (я_могу_подключиться_к_mysql_c("хост.вашей.базы_данных",
                                    "ваше-имя_пользователя", "ваш-пароль")
    переход_ко_всему_что_есть_в_базе_данных();
  else
    отправка_ошибки_пользователю_с_использованием_die
?>
```

Но такой код требует длительного набора, поэтому PHP позволяет сократить его до следующего вида:

```
<?php
  mysql_connect("хост.вашей.базы_данных",
                "ваше-имя_пользователя", "ваш-пароль")
    or die("<p>Ошибка подключения к базе данных: " .
           mysql_error() . "</p>");

  echo "<p>Вы подключились к MySQL!</p>";
?>
```

Тем самым не только сокращается сценарий, но и решаются некоторые другие вопросы. По сути, здесь говорится, что нужно попытаться подключиться (используя mysql_connect), и если возвращенный результат не будет иметь значение true (часть кода после слова or), нужно прекратить работу, — die. Инструкция die выводит сообщение об ошибке, но она еще и устраивает «смерть». Иными словами, она завершает работу сценария. Итак, если mysql_connect возвращает false и запускается инструкция die, происходит выход из вашего сценария. Ваши пользователи

уже не увидят строку «Вы подключились к MySQL!», потому что работа сценария прекратится. Он умрет на полу комнаты, в которой стоит сервер, в поисках работоспособного подключения к базе данных. (Подробности инструкции die рассмотрены во врезке «Курсы повышения квалификации. Все когда-нибудь умирают» в следующем подразделе.)

Но это еще не все. Инструкция mysql_connect при невозможности подключения задействует еще одну функцию. Она делает ошибку, с которой столкнулась при попытке подключения, доступной для другой инструкции — mysql_error. Следовательно, вы можете вызвать mysql_error в качестве части инструкции die, чтобы показать, что произошло на самом деле.

ПРИМЕЧАНИЕ

С технической точки зрения mysql_connect, mysql_error и die являются примерами функций. Функция — это блок кода, которому обычно присваивается имя и который можно вызвать из своего собственного кода в любое время, когда понадобится этот блок. Вызывать функцию по имени значительно быстрее и лучше, чем каждый раз переписывать блок кода, который представляет эта функция.

Но пока не стоит волноваться насчет функций. Просто используйте их как любые прежние PHP-команды. Со временем вы не только лучше поймете, что такое функции, но и станете создавать собственные.

Если инструкция mysql_connect сумела создать подключение без всяких проблем, она возвращает это подключение. PHP пропустит строку с инструкцией die, а затем выполнит следующую строку:

```
echo "<p>Вы подключились к MySQL!</p>";
```

Чтобы посмотреть эту команду в действии, создайте простую HTML-форму и назовите ее connect.html.

Для начала можно воспользоваться следующим HTML:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пример 4.1</div>

  <div id="content">
    <h1>Тест подключения к SQL</h1>
    <form action="scripts/connect.php" method="POST">
      <fieldset class="center">
        <input type="submit" value="Подключиться к MySQL" />
      </fieldset>
    </form>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

Этот сценарий из разряда проще не придумаешь: он создает форму, вставляет в нее одну-единственную кнопку, которую привязывает к новому сценарию `connect.php`. Загрузите форму в браузер (рис. 4.1) и щелкните на кнопке Подключиться к MySQL. Конечно, можно сделать `connect.html` еще проще. Можно убрать все ссылки на структуру и на CSS. Но кому захочется подключаться к базе данных с какой-то невзрачной страницы? Клиентам нравятся красивые и аккуратные сайты. Не нужно тратить слишком много времени на CSS. Придайте профессиональный вид хотя бы самым основным демонстрационным сайтам, и клиенты станут относиться к вам куда почтительнее.

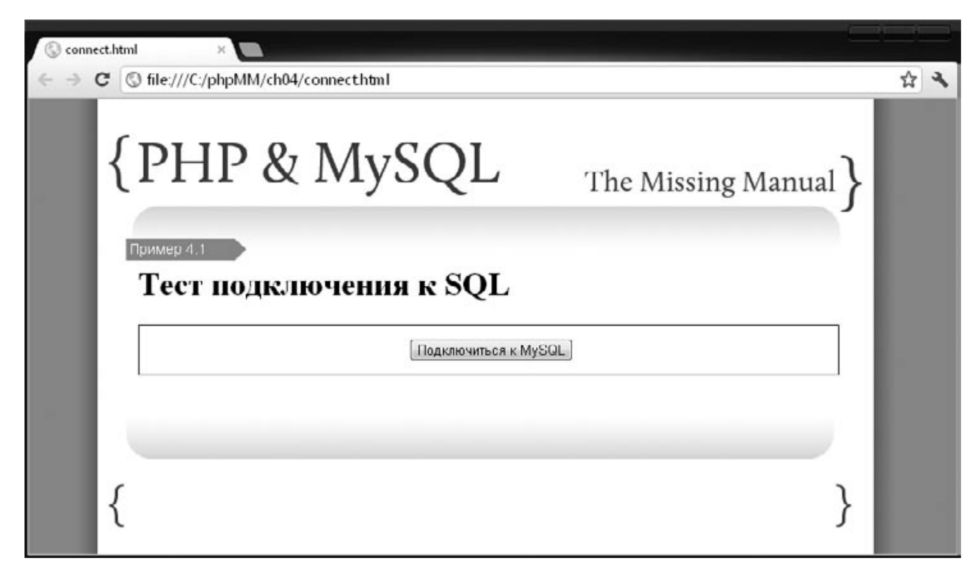


Рис. 4.1. Форма подключения к MySQL

Будем надеяться, что вы увидите одно из самых приятных сообщений для начинающего программиста: вы подключились! Чтобы насладиться успехом, убедитесь, что у вас на экране то же самое, что и на рис. 4.2.



Рис. 4.2. Подключение к MySQL установлено

Эти несколько слов означают, что теперь ваш PHP-сценарий может делать с вашей базой данных практически все, что только можно себе представить. Но здесь кое-чего не хватает: как MySQL узнает, которая база данных ваша? Вам нужно еще сообщить PHP, какой базой следует пользоваться.

Выбор используемой базы данных

А теперь упомяну об одной замечательной особенности, которая ждет программистов: почти все функции семейства `mysql_` работают одинаково: вы даете им некие значения, а они возвращают что-либо полезное. Если происходит что-нибудь нехорошее, то вы обычно получаете либо значение `false`, либо несуществующий объект (нечто, называемое большинством программистов `null` или `nil`).

Итак, теперь вам нужно сказать MySQL, какой базой данных хочет воспользоваться ваш PHP-сценарий. Для этого есть специальная функция `mysql_select_db`.

ПРИМЕЧАНИЕ

Семейство mysql_ состоит из множества функций. Вы можете сделать закладку на странице документации этого семейства: www.php.net/manual/ru/ref.mysql.php. Если вы когда-либо попадете в тупиковую ситуацию, загляните туда и посмотрите, может ли функция выполнить задуманное вами действие.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Все когда-нибудь умирают

Легкость, с которой можно забыть добавить инструкции `die` к PHP-сценариям, весьма настораживает. PHP не требует применения этих инструкций и вполне готов принять код, имеющий следующий вид:

```
mysql_connect("хост_вашей_базы_данных",
    "имя_пользователя", "пароль");
```

Это такой же код, как уже использовавшийся вами, но в нем нет части, которая относится к инструкции `die`.

И вот что получается: когда `die` отсутствует и при этом что-нибудь идет не так, сценарий потерпит аварию и предоставит нечто такое, что будет либо абсолютно бесполезным сообщением об ошибке, либо столь непонятным, что вы даже не сможете дать этому внятное определение. Например, если вы не воспользуетесь инструкцией `die` и, запустив сценарий, введете неверный пароль, то вы получите примерно следующее сообщение об ошибке:

```
Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)
```

Вообще, это сообщение об ошибке еще достаточно информативное, по сравнению с тем, что обычно происходит, если не применить инструкцию `die`. Добавление строки обработки ошибки может внести существенные изменения в то, что получает пользователь в случае каких-либо сбоев.

⇗

Фактически, когда вы начнете создавать куда более объемные и полноценные веб-приложения, вы сможете перенаправлять пользователя на красиво отформатированную страницу сообщения об ошибке, имеющую контактную информацию для связи с администратором и оформленный с применением CSS-стилей отчет об ошибке. Но все это невозможно без применения инструкции `die`.

Когда вы освоите PHP получше и будете считать себя профессионалом, вы можете решить, что ошибки — довольно редкое явление. Вы можете подумать, что инструкция `die` относится к арсеналу любителей, не способных создавать безупречный код. К сожалению, когда вы в два часа ночи пытаетесь завершить какой-нибудь этап своей работы, без которого не получите зарплату, ваш уровень профессионализма скатывается до любительского и сообразительность резко снижается. Никто не застрахован от ошибок, и инструкция `die` (наряду с другими технологиями обработки ошибок) является одним из средств спасения, которые в случае совершения неизбежных ошибок помогают вам сохранить авторитет подготовленного и профессионального специалиста.

Кстати, самые квалифицированные и высокооплачиваемые программисты мира являются признанными гуру в области обработки ошибок. В то же самое время они могут и не пользоваться инструкцией `die`. Скорее всего, они применяют более надежные системы обработки ошибок, подобные той, которую вы будете использовать в главе 7. Но пока применение инструкции `die` поможет вам привыкнуть к добавлению той или иной формы обработки ошибок.

Функции `mysql_select_db` передается имя базы данных, и, как вы уже догадались, данная функция применяет к этому имени команду `USE` или возвращает значение `false`. Поэтому чтобы использовать нужную базу данных, следует обновить сценарий в файле `connect.php`:

```
<?php
  mysql_connect("хост_вашей_базы_данных",
                "ваше_имя_пользователя", "ваш_пароль")
    or die("<p>Ошибка подключения к базе данных: " .
          mysql_error() . "</p>");

  echo "<p>Вы подключились к MySQL!</p>";

  mysql_select_db("имя_вашей_базы_данных")
    or die("<p>Ошибка при выборе базы данных bmclaugh: " .
          mysql_error() . "</p>");

  echo "<p>Вы подключены к MySQL с использованием базы данных bmclaugh.</p>";
?>
```

Вы уже видели эту схему. Инструкция `die` гарантирует, что в аварийной ситуации сценарий выведет отчет об ошибке, который сможет прочитать пользователь, а затем завершит свою работу. Если все пройдет удачно, будет выведено другое, более оптимистичное сообщение.

Испытайте новую версию в работе. Посетите еще раз страницу connect.html и попробуйте воспользоваться вашей базой данных и подключиться к ней (теперь уже с помощью команды USE). Желаемый результат показан на рис. 4.3. Следующий шаг: общение с вашей базой данных с помощью SQL.

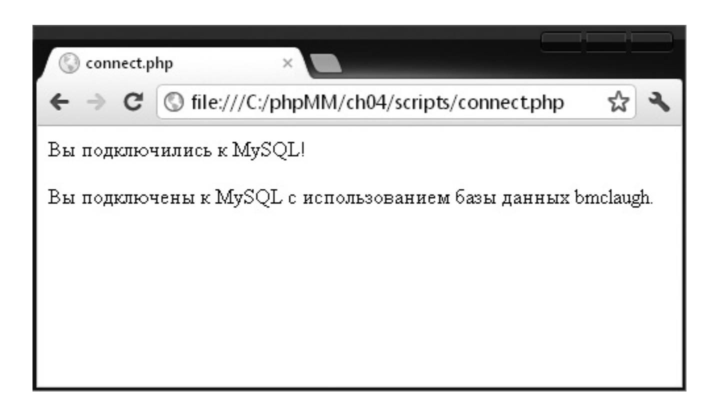


Рис. 4.3. У сценария есть подключение к MySQL, и он использует нужную базу данных

Демонстрация таблиц базы данных с помощью команды SHOW

После подключения и привязки к нужной базе данных вы должны увидеть, с чем именно можно работать. Одним из первых действий, совершенных с помощью окна командной строки MySQL, был просмотр перечня существующих таблиц, а затем создание собственных таблиц. При создании таблиц придется приложить немного больше усилий. Вы займетесь этим с помощью новой HTML-формы и нового сценария.

Но перед погружением в эту работу можно без особого труда обзавестись сценарием для просмотра таблиц, доступных в вашей базе данных. Еще раз откройте файл connect.php и добавьте в него следующую строку:

```
<?php
  // Весь ваш код подключения к существующей базе данных

  $result = mysql_query("SHOW TABLES;");
?>
```

Здесь используется еще одна новая для вас функция перехода от PHP к MySQL: mysql_query. Вы должны хорошенько изучить эту функцию, поскольку она является ключевой в процессе передачи SQL в вашу базу данных. Этой функции передается SQL, и в данном случае вы передаете ей совсем простой SQL-код:

```
SHOW TABLES;
```

Данная команда точно соответствует набору SQL-кода в окне командной строки.

Обработка ошибок путем наблюдения за отсутствием результата

А как же насчет использования инструкции die? Как насчет обработки ошибок? В приведенном выше коде ничего этого нет, но теперь-то вы знаете, что все это должно быть. Там все делается несколько иначе: все, что возвращается функцией mysql_query, помещается в переменную по имени $result. И это тот самый результат, который нужно проверить. В нем будет содержаться либо перечень таблиц, возвращенный командой SHOW TABLES, либо какой-нибудь отчет об ошибке. И если там будет отчет об ошибке, переменная $result будет иметь значение false, поскольку, сталкиваясь с проблемами, функции семейства mysql_ возвращают false.

Вы уже знаете, как проверять на наличие значения false, и можете добавить этот код для обработки возникающих проблем:

```
<?php
  // Весь код подключения к существующей базе данных

  $result = mysql_query("SHOW TABLES;");

  if ($result === false) {
    die("<p>Ошибка при выводе перечня таблиц: " . mysql_error() . "</p>");
  }
?>
```

Хотя это вполне рабочий код, но в действительности многие PHP-программисты так не делают. Оператор === используется в PHP крайне редко, по крайней мере для того, чтобы проверить, не имеет ли переменная значения false. Чаще всего в PHP это выполняется по-другому, с использованием оператора отрицания, который еще называется оператором восклицательного знака: !. Поэтому, если нужно проверить, не имеет ли переменная по имени $some-variable значения false, это можно выразить следующим образом: if (!$some-variable). Знак ! скажет: «Посмотрим, не имеет ли $some-variable значение false».

Еще лучше думать о знаке ! как о слове not (не). Итак, фактически вы хотите выразить в своем коде следующее: if not $result, then die (если значение переменной $result не равно true, завершить выполнение сценария). В соответствии с этим можно переписать код, придав ему следующий вид:

```
<?php
  // Весь код подключения к существующей базе данных

  $result = mysql_query("SHOW TABLES;");

  if (!$result) {
    die("<p>Ошибка при выводе перечня таблиц: " . mysql_error() . "</p>");
  }
?>
```

Этот вариант кода с точки зрения PHP выглядит намного лучше, и теперь в нем решены все проблемы.

ПРИМЕЧАНИЕ

Фраза «именно так это делается в PHP» может звучать довольно странно. Раз код работает, значит, его можно применять, не так ли? Конечно, так... но вы слышали, как говорит человек, недавно приступивший к изучению родного для вас языка? Зачастую он говорит правильные слова, но при этом произносит их неверно, использует неправильный порядок их следования и путает контекст их применения.

Языки программирования в этом смысле похожи на обычные языки. Можно написать работоспособный код, а можно создать код, свидетельствующий о вашем знании языка. Иногда это называется выразительностью. Стоит учиться не только написанию работоспособного PHP-кода, но и созданию такого кода PHP, который имеет естественный вид.

Чтобы убедиться в том, что код справляется с ошибками, измените свой SQL-запрос, допустив в нем опечатку:

```
<?php
  // Весь код подключения к существующей базе данных

  $result = mysql_query("SHOWN TABLES;");

  if (!$result) {
    die("<p>Ошибка при выводе перечня таблиц: " . mysql_error() . "</p>");
  }
?>
```

Теперь загрузите в браузер файл `connect.html` и запустите свой тест подключения (рис. 4.4). Результат будет по-прежнему немного непонятным, но дающим понять, что код обнаружил проблему и обработал ее, выдав сообщение об ошибке, а не какой-нибудь массивный сплав непонятной информации.

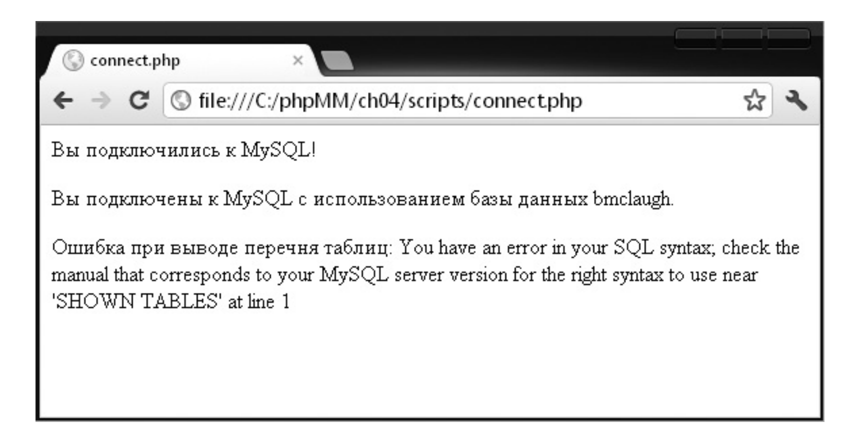


Рис. 4.4. Результат обработки кода с ошибкой

Вывод SQL-результатов

Ошибки обрабатываются, отчеты о проблемах выводятся, теперь можно в конце концов заняться текущим результатом — тем, что на самом деле находится в переменной `$result`, когда все обходится без отказов. К сожалению, в данном случае

обстоятельства складываются несколько сложнее. Значение переменной `$result` не относится к какому-нибудь из уже использованных вами PHP-типов или к какому-нибудь типу, непосредственной работе с которым нужно научиться. Значение этой переменной называется *ресурсом*, что на языке PHP является понятием для специальной переменной, относящейся к чему-нибудь, что находится за пределами PHP.

Нужно представлять себе это следующим образом: в случае использования `mysql_query` вы запрашиваете SQL-результаты путем запуска запроса `SHOW TABLES`. Но наряду с тем, что PHP может передавать информацию MySQL, этот язык не знает, как интерпретировать SQL. Поэтому он не может знать, что в переменной `$result` должен храниться перечень строк, в каждой из которых будет одно значение: имя таблицы. Он знает только, что нечто другое, а именно ваша база данных, получает запрос благодаря функции `mysql_query`. И полагает следующее: в зависимости от того, какой именно запрос был передан `mysql_query`, в переменной `$result` могут содержаться строки с несколькими фрагментами информации, такими как имя и URL-адрес в Facebook или просто свидетельство о том, сработала или нет инструкция `CREATE TABLE`.

Итак, в таких случаях обычно приходится иметь дело с PHP-ресурсом. Этот ресурс что-то означает; то, о чем PHP не знает, что это такое. Поэтому ваш PHP нуждается в помощи. Ему нужно нечто, разбирающееся в MySQL и способное определить, как нужно работать с переменной `$result`. Именно это можно получить с помощью другой MySQL-функции — `mysql_fetch_row`. Ей передается ресурс, возвращенный функцией `mysql_query`, которая позволяет осуществить последовательный перебор строк в результате, возвращенном вашим SQL-запросом.

Основная схема имеет следующий вид.

1. Напишите свой SQL-запрос и сохраните его в строке или в переменной.
2. Передайте свой запрос в `mysql_query` и получите PHP-ресурс.
3. Передайте этот ресурс в `mysql_fetch_row`, чтобы получить в ответ результат в построчном виде.
4. Организуйте циклический перебор этих строк и извлеките нужную вам информацию.
5. Купите на все заработанные деньги по-настоящему красивую гитару.

ПРИМЕЧАНИЕ

Последнее действие является необязательным, но настоятельно рекомендуемым.

У вас есть ресурс, хранящийся в переменной `$result`. Следовательно, теперь его нужно передать в функцию `mysql_fetch_row`:

```
<?php
  // Весь код подключения к существующей базе данных

  $result = mysql_query("SHOW TABLES;");

  if (!$result) {
```

```
    die("<p>Ошибка при выводе перечня таблиц: " . mysql_error() . "</p>");
  }

  echo "<p>Таблицы, имеющиеся в базе данных:</p>";
  echo "<ul>";
  while ($row = mysql_fetch_row($result)) {
    // Какие-либо действия с содержимым переменной $row
  }
  echo "</ul>";

?>
```

ВНИМАНИЕ

Если вы выше заменяли в коде SHOW TABLES на SHOWN TABLES, чтобы получить ранее рассмотренную ошибку, не забудьте внести обратные изменения и вернуть работоспособность коду SQL.

Даже если PHP не знает, что делать с ресурсом, возвращенным функцией mysql_query, это знает функция mysql_fetch_row. Она получает ваш ресурс $result и начинает разбивать строки на элементы массива.

А затем следует также новый для вас цикл while, в смысле которого вы, возможно, уже разобрались. Цикл while продолжается до тех пор, пока управляющее им выражение имеет значение true. В данном случае цикл продолжается, пока переменная $row, хранящая очередной результат, который получен от вашего SQL-запроса, получает значение из выражения mysql_fetch_row($result). Когда строк в результате больше нет, функция mysql_fetch_row ничего не возвращает, поэтому $row имеет пустое значение и цикл while говорит: «Ну вот, я все сделал и теперь прекращаю циклический перебор».

В результате вы получите красивый ненумерованный список (ul), готовый показать по отдельности каждую строку, к которому осталось добавить кое-что еще:

```
<?php
  // Весь код подключения к существующей базе данных

  $result = mysql_query("SHOW TABLES;");

  if (!$result) {
    die("<p>Ошибка при выводе перечня таблиц:" . mysql_error() . "</p>");
  }

  echo "<p>Таблицы, имеющиеся в базе данных:</p>";
  echo "<ul>";
  while ($row = mysql_fetch_row($result)) {
    echo "<li>Таблица: {$row[0]}</li>";
  }
  echo "</ul>";

?>
```

Этот код должен также быть для вас знакомым. При каждом возвращении значения переменной `$row` с помощью функции `mysql_fetch_row` происходит возвращение массива, то есть структуры данных, которая уже рассматривалась в подразделе «Массивы могут содержать несколько значений» раздела «Переменная $_REQUEST» главы 2. В этом массиве содержится все многообразие той информации, которая была возвращена благодаря вашему SQL-запросу. Для запроса `SHOW TABLES` там содержится только один фрагмент информации: имя таблицы в элементе `$row[0]`. Совсем скоро вы начнете создавать более сложные запросы, и тогда может потребоваться использовать значения, хранящиеся в `$row[1]`, в `$row[2]` или даже в `$row[10]`.

Итак, в данном случае при получении `$row` извлекается имя таблицы путем получения первого элемента массива с индексом `0`, а затем его значение выводится с помощью инструкции `echo`. Однако здесь есть один непонятный момент: зачем здесь фигурные скобки внутри строки, передаваемой инструкции `echo`?

Эту строку можно переписать следующим образом:

```
while ($row = mysql_fetch_row($result)) {
  echo "<li>Таблица: " . $row[0] . "</li>";
}
```

Тут все понятно, за исключением всех этих дополнительных кавычек и точек, позволяющих объединить строки.

ПРИМЕЧАНИЕ

Специалисты могли бы вам напомнить, что объединение строк называется конкатенацией (см. подраздел «Объединение текста» раздела «Работа с текстом в PHP» главы 2).

Но PHP весьма продуманный язык, а его создатели к тому же еще и программисты. Они, как и вы, посчитали, что вам постоянно приходится вставлять значения переменных в середину строк. Поэтому вместо того, чтобы постоянно закрывать строку и добавлять переменную, можно просто заключить переменную в фигурные скобки `{ }`, и PHP выведет значение этой переменной, а не строковое значение `"$row[0]"`. Это сделано для упрощения кода и является весьма удобным приемом программирования.

Сохраните файл `connect.php`, загрузите еще раз файл `connect.html` в браузер и посмотрите, какие таблицы доступны в вашей базе данных. На рис. 4.5 показан результат работы сценария `connect.php`, запущенного в отношении базы данных, которая содержит большое количество таблиц. Вам может хватить всего одной или двух таблиц. Нужно просто убедиться в том, что вы видите перечень ваших таблиц.

ПРИМЕЧАНИЕ

Довольно скоро станет практически бесполезно использовать в браузерах команду SHOW TABLES. Результат может выглядеть как громадный перечень таблиц, ведь вполне вероятно, что в создаваемых вами приложениях будут задействованы 20, 30 или даже 100 таблиц. Но в данный момент применение этой команды является весьма простым способом убедиться, что ваши PHP-сценарии общаются с вашими базами данных MySQL.



Рис. 4.5. Результат работы сценария connect.php

Приведение кода в порядок с помощью нескольких файлов

Если вы еще не поняли, у вашего сценария `connect.php` есть ряд узких мест. Посмотрите на первые несколько вызовов MySQL:

```
<?php
  mysql_connect("хост_вашей_базы_данных",
                "ваше_имя_пользователя", "ваш_пароль")
    or die("<p>Ошибка подключения к базе данных: " .
           mysql_error() . "</p>");

  echo "<p>Вы подключились к MySQL!</p>";

  mysql_select_db("имя_вашей_базы_данных")
    or die("<p>Ошибка при выборе базы данных bmclaugh: " .
           mysql_error() . "</p>");

  echo "<p>Вы подключены к MySQL с использованием базы данных bmclaugh.</p>";

  // И так далее...
?>
```

Вы вручную набираете в своем сценарии имя хоста своей базы данных, свое имя пользователя, свой пароль и имя своей базы данных. А теперь предположим, что у вас имеется 10 сценариев и вы набираете их имена 10 раз. Довольно высока вероятность опечатки.

Но дело не только в этом. Что произойдет, когда вы смените пароль? Или что будет, если вы перейдете на более совершенный хостинг, чтобы справиться со всем веб-трафиком, генерируемым вашими приложениями, что вызовет необходимость изменить имя хоста вашей базы данных? Вам придется проследить все места, куда помещена соответствующая информация в каждом PHP-сценарии. Это превратится в ночной кошмар, не давая вам заняться новым кодом и повысить свой заработок. Незавидная перспектива.

Вам нужен способ *абстрагирования* от таких фрагментов информации. Абстрагирование является понятием в программировании, которое означает скрытие реализации, то есть способа работы какого-нибудь компонента (например, пароля) от программ, которые его используют. Обычно у вас есть обозначение или имя, и это имя ссылается на что-нибудь еще, имеющее значительно больше деталей. И даже если эти детали изменятся, имя все равно будет указывать на нужные компоненты.

Это все равно, что сказать «Ли», подразумевая при этом мою жену и не испытывая необходимости говорить «эта эффектная длинноногая блондинка с короткой стрижкой 34 лет от роду». И вся прелесть обращения «Ли» состоит в том, что каждый день рождения вы можете продолжать говорить «Ли», вместо того чтобы менять свое описание.

Замена набранных вручную значений переменными

Представьте, что вам захотелось, чтобы ваш код стал похож на следующий (а вам действительно этого захочется):

```
<?php
  mysql_connect($database_host, $username, $password)
    or die("<p>Ошибка подключения к базе данных: " .
           mysql_error() . "</p>");

  echo "<p>Вы подключились к MySQL!</p>";
  mysql_select_db($database_name)
    or die("<p>Ошибка при выборе базы данных bmclaugh: " .
           mysql_error() . "</p>");

  echo "<p>Вы подключены к MySQL с использованием базы данных bmclaugh. </p>";

  // И так далее...
?>
```

Вы всего лишь вместо ручного набора имени пользователя и имени базы данных записываете в код что-то похожее на переменную. Теперь вы можете определить такие переменные чуть выше вашего кода подключения:

```
<?php
  $database_host = "хост.вашей.базы_данных";
  $username = "ваше-имя_пользователя";
  $password = "ваш-пароль";
  $database_name = "имя-вашей-базы-данных";

  // Код подключения к базе данных
?>
```

Но действительно ли все это намного лучше прежнего кода? Еще нет, поскольку вам по-прежнему приходится вручную набирать те же самые значения в своем сценарии. Вам нужно убрать эти значения в файл, чтобы исключить их ручной набор. Читаем дальше.

Абстрагирование важных значений путем заключения их в отдельный файл

Ваша цель заключается в выведении указанных значений из файла connect.php в какое-нибудь другое место (к которому смогут иметь доступ все ваши сценарии PHP), чтобы избавиться от ручного ввода. Создайте новый файл и назовите его app_config.php. Теперь переместите свои переменные в этот новый файл:

```
<?php
// Константы подключения к базе данных
$database_host = "хост.вашей.базы_данных";
$username = "ваше-имя_пользователя";
$password = "ваш-пароль";
$database_name = "имя-вашей-базы-данных";

?>
```

ПРИМЕЧАНИЕ

Сохраните app_config.php в такое место, чтобы оно было доступно всем сценариям вашего приложения, обращающимся к этому файлу. В примерах, разработанных для этой книги, файл app_config.php находится в корневом каталоге сайта в подкаталоге scripts/. Поэтому, если вы находитесь в каталоге ch04/scripts/, доступ к этому файлу можно получить, используя путь ../../scripts/app_config.php или [корневой_каталог_сайта]/scripts/app_config.php. Файл можно сохранить где угодно, если указать путь к нему непосредственно в вашем PHP-сценарии, который на него ссылается.

При переходе на эксплуатационную версию своего приложения вам, скорее всего, потребуется поместить этот файл где-нибудь за пределами корневого каталога сайта. Тогда веб-пользователи не смогут просто набрать путь к вашему конфигурационному сценарию и заполучить все ваши пароли. Кроме того, вы можете

повысить безопасность этого каталога, хотя обычно легче всего просто убрать его из каталогов, которые попадают в поле зрения веб-обслуживания.

Теперь все многообразие ваших PHP-сценариев может пользоваться этими общими переменными. Измените переменную в файле app_config.php, и это изменение повлияет на все ваши PHP-сценарии, пользующиеся этими общими переменными.

Но как получить доступ к этим переменным? Вернитесь к файлу connect.php и удалите те места, где вы определяли эти переменные. Однако если сейчас попытаться получить доступ к connect.php посредством connect.html, появится весьма неприятное сообщение об ошибке (рис. 4.6).

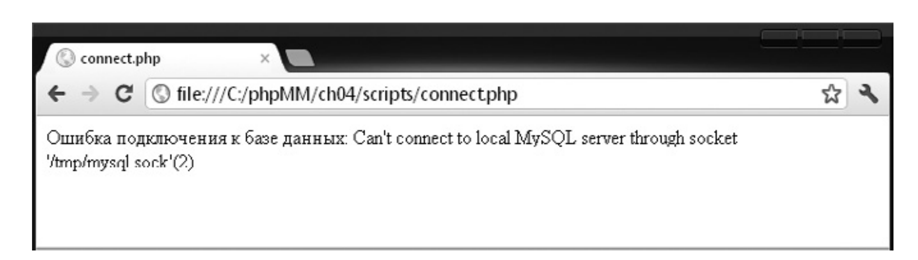


Рис. 4.6. Сообщение об ошибке

Возникновение ошибки обусловлено тем, что connect.php пока не знает, на что ссылается переменная $username или $password. Вам нужно проинформировать PHP, что перед попыткой совершать какие-либо действия в connect.php требуется загрузить файл app_config.php. И практически именно это вы и вводите в свой сценарий:

```
<?php

  require '../../scripts/app_config.php';

  // Код подключения к базе данных
?>
```

Теперь PHP загружает файл ../../scripts/app_config.php перед тем, как запускает вашу функцию mysql_connect. По сути, инструкция require говорит: «Эй, PHP, если ты не можешь загрузить указанный мною файл, то выдай ошибку, поскольку ничего больше работать не будет».

ВНИМАНИЕ

Убедитесь, что передаваемые инструкции require путь и имя файла соответствуют тому месту, куда действительно был помещен файл app_config.php. В противном случае вы получите ошибку, выданную инструкцией require.

Еще раз попробуйте запустить сценарий подключения. Вы должны увидеть перечень своих таблиц, что будет говорить о восстановлении работоспособности сценария.

ПОД КАПОТОМ

Require или Include?

В PHP есть еще одна очень похожая на `require` инструкция — `include`. Она точно так же, как и `require`, отдает PHP команду загрузить еще один файл. Разница заключается в том, что если этот файл не может быть загружен, `include` просто выдает предупреждение и позволяет PHP продолжить выполнение всех следующих команд, имеющихся в сценарии. Иными словами, `require` полностью прекращает работу сценария, а `include` дает возможность продолжить его выполнение.

И тут возникает вопрос. Нужно ли вообще связываться с включением файла, если он не нужен? В большинстве случаев, скорее всего, нет. И вы включаете этот файл, потому что он вам нужен, вы требуете (require) запуска этого файла. Поэтому практически в любой ситуации для получения другого файла нужно использовать `require`, а не `include`. Если что-нибудь пойдет не так, вам нужно быть в курсе случившегося. Вам не нужно, чтобы запускался весь остальной код, поскольку это, скорее всего, в любом случае приведет к выдаче ошибки.

Переменные изменяются, а константы сохраняют постоянство

В нашем коде осталась еще одна небольшая проблема: для вашего имени пользователя и пароля, а также для имени хоста базы данных и ее имени по-прежнему используются переменные. А что такое переменная? Это то, что подвержено изменениям. Соответственно, PHP беспрепятственно позволяет написать в `connect.php` следующий код:

```
mysql_connect($database_host, $username, $password)
  or die("<p>Ошибка подключения к базе данных: " . mysql_error() . "</p>");

// Допускается и такой вариант
$password = "hijinks";
```

Итак, что произойдет, когда какой-нибудь другой сценарий, который также требует загрузки файла `app_config.php`, пытается подключиться с помощью `mysql_connect`? Он собирается воспользоваться переменной `$password`, но она больше не содержит правильного значения. Ей присвоено значение `"hijinks"`, что вызовет сплошной хаос.

На самом деле вам нужно, чтобы эти значения в файле `app_config.php` были константами и никогда не изменялись. Это можно сделать с помощью специальной функции `define`. Откройте файл `app_config.php` и внесите изменения в ваш код:

```
<?php
// Константы подключения к базе данных
define("DATABASE_HOST", "хост_вашей_базы_данных");
```

```
define("DATABASE_USERNAME", "ваше_имя_пользователя");
define("DATABASE_PASSWORD", "ваш_пароль");
define("DATABASE_NAME", "имя_вашей_базы_данных");
?>
```

Вы определяете имя константы и ее значение, а PHP создает новую константу. Благодаря этому вы можете набрать в своем коде DATABASE_HOST, а PHP на самом деле увидит "хост_вашей_базы_данных". Отлично! И поскольку это константа, она не может быть изменена в какой-нибудь строке кода.

Имена констант набираются только заглавными буквами. Сами по себе заглавные буквы не нужны, но это еще один способ «разговаривать в манере PHP-программиста». Нужно, чтобы константы отличались от переменных, и использование заглавных букв является одним из способов получения такого результата. У констант также нет знака $ перед их именем, что является еще одним способом отличить их от переменных.

Теперь нужно внести небольшие изменения в файл connect.php, чтобы воспользоваться новыми именами констант, состоящими из одних заглавных букв:

```
<?php
  require '../../scripts/app_config.php';

  mysql_connect(DATABASE_HOST, DATABASE_USERNAME, DATABASE_PASSWORD)
    or die("<p>Ошибка подключения к базе данных: " .
           mysql_error() . "</p>");

  echo "<p>Вы подключились к MySQL!</p>";

  mysql_select_db(DATABASE_NAME)
    or die("<p>Ошибка при выборе базы данных " . DATABASE_NAME .
           mysql_error() . "</p>");

  echo "<p>Вы подключены к MySQL с использованием базы данных " . DATABASE_NAME .
"</p>";

  // Продолжение действий, инициируемых SQL-запросом...
?>
```

ВНИМАНИЕ

Использовать фигурные скобки для вывода значения константы внутри кавычек нельзя. PHP выводит значение только в том случае, когда в фигурные скобки заключается переменная (чье имя начинается со знака $). Вместо этого для вывода значения константы нужно воспользоваться обычной конкатенацией строк с закрытием строки и добавлением константы с помощью точки (.), как описано в подразделе «Объединение текста» раздела «Работа с текстом в PHP» главы 2.

Попробуйте еще раз запустить connect.php. Вы должны увидеть красивый перечень имен таблиц. И на этот раз для вашей важной информации вы получили константы, убранные из соображений безопасности в файл, выделенный из connect.php.

ПРИМЕЧАНИЕ

Было бы также неплохо повысить безопасность файла app_config.php и любых других сценариев, в которых содержатся специальные значения вроде паролей. Можно дополнительно ограничить права доступа к файлу или переместить файл туда, где к нему может обратиться ваш PHP-сценарий, но не смогут обратиться веб-пользователи. Если вы не знаете, как это сделать, попросите администратора сети или сервера помочь вам.

ЗАМЕТКИ О ПРОЕКТИРОВАНИИ

Начните с малого, добавляйте понемногу, заканчивайте малым

У вас может возникнуть вопрос, почему нельзя было сразу начать с `app_config.php` и с полностью готовой работоспособной версии `connect.php`. Или как минимум почему сразу не поместить весь код подключения к базе данных в `connect.php`, а затем не поместить туда сразу весь код вывода на экран? Разве не так создают свой код настоящие разработчики?

И да, и нет. Многие разработчики примерно так и пишут свой код. Они набирают 10, 20, 50 строк кода в своем сценарии, а затем испытывают его в работе. При этом будет много отказов, поскольку разработчики набирали код очень быстро и допускали ошибки. Затем они поэтапно устраняют каждую проблему. И многих разработчиков это устраивает.

Однако это не самый эффективный способ работы. Кроме того, основное внимание обычно уделяется самому последнему этапу (вроде вывода таблиц), и поэтому можно потратить слишком мало времени на выявление наилучшего способа обработки информации на промежуточных этапах. Можно упустить возможность использования фигурных скобок `{}` для упрощения инструкции, которая выводит `$row[0]`, или можно пропустить инструкцию `die`, поскольку мысли были сосредоточены на выводе HTML, а не на обработке того случая, когда пароль базы данных окажется неверным.

Лучшие разработчики работают поэтапно с действительно весьма и весьма небольшими порциями кода. Они тестируют этот код, а затем переходят к чему-нибудь другому. Множество по-настоящему первоклассных разработчиков сначала пишут тесты, а уж потом занимаются всем остальным. Они пишут эти тесты, и тесты, конечно же, проваливаются, поскольку код ими еще не написан. Затем они пишут код, достаточный для прохождения их теста, а затем они пишут следующий тест.

Поначалу в этом методе можно не уловить никакого смысла. Зачем создавать тесты для несуществующего кода? А вот и вовсе форменный идиотизм: зачастую данный подход выливается в то, что объем кода теста превышает объем кода самого приложения!

Это большой объем работы, и он основан на идее, что на протяжении одного этапа нужно создавать достаточный объем кода, чтобы получить какое-нибудь одно работоспособное действие.

Но открою вам большую тайну, почему первоклассные разработчики считаются таковыми: подход, при котором сначала создаются тесты, позволяет получать наилучший код. Работа, разбитая на небольшие этапы от начала до конца, сконцентрирована на каком-нибудь одном вопросе и позволяет довести его до совершенства. Они не спешат перейти к чему-нибудь другому. И это означает, что предмет их труда надежен и работоспособен. Этот подход поначалу занимает больше времени, но в результате получается очень надежный код, который впоследствии практически не создает аварийных ситуаций.

Поэтому запаситесь терпением и работайте поэтапно малыми долями. Ваш код станет лучше, и вы заработаете авторитет в глазах заказчиков. Потому что ваш код как работал, так и работает, а они сидят на телефоне, пытаясь добиться помощи в восстановлении работы приложения, разработанного «другими ребятами».

Создание элементарного исполнителя SQL-запросов

Теперь, когда вы уже можете подключаться к SQL, можно приступить к решению и более амбициозной задачи: созданию своего собственного средства командной строки MySQL. Разумеется, теперь вы — PHP-разработчик и программист, поэтому нужно мысленно вычеркнуть из определения этого средства слова о командной строке и вписать туда слова о том, что это средство будет основано на применении веб-технологий.

У вас уже есть основная часть необходимого для этого инструмента. Вам совсем нетрудно будет создать HTML-форму, позволяющую вам и вашим пользователям вводить SQL-запрос, вы знаете, как подключиться к MySQL и выбрать базу данных, и вы можете запустить запрос. Осталось только определить, как интерпретировать PHP-ресурс, который возвращает функция mysql_query, когда это не список имен таблиц.

Создание HTML-формы с большим пустым полем ввода

Но перед тем как переходить к функции mysql_query и ее результатам, начнем с того, что нам уже известно: с HTML-формы. Пока не нужно ничего усложнять, необходимо просто создать форму с одним полем для ввода многострочного текста, где будут набираться запросы, и несколькими основными кнопками.

Откройте свой текстовый редактор и создайте файл queryRunner.html:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
```

```
  <div id="example">Пример 4.2</div>

  <div id="content">
    <h1>Средство запуска SQL-запросов</h1>
    <p>Введите свой SQL-запрос в это поле:</p>
    <form action="scripts/run_query.php" method="POST">
      <fieldset>
        <textarea id="query_text" name="query"
                  cols="65" rows="8"></textarea>
      </fieldset>
      <br />
      <fieldset class="center">
        <input type="submit" value="Запуск запроса" />
        <input type="reset" value="Очистка и перезапуск" />
      </fieldset>
    </form>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

Запустите свой браузер и убедитесь, что все выглядит так, как на рис. 4.7.

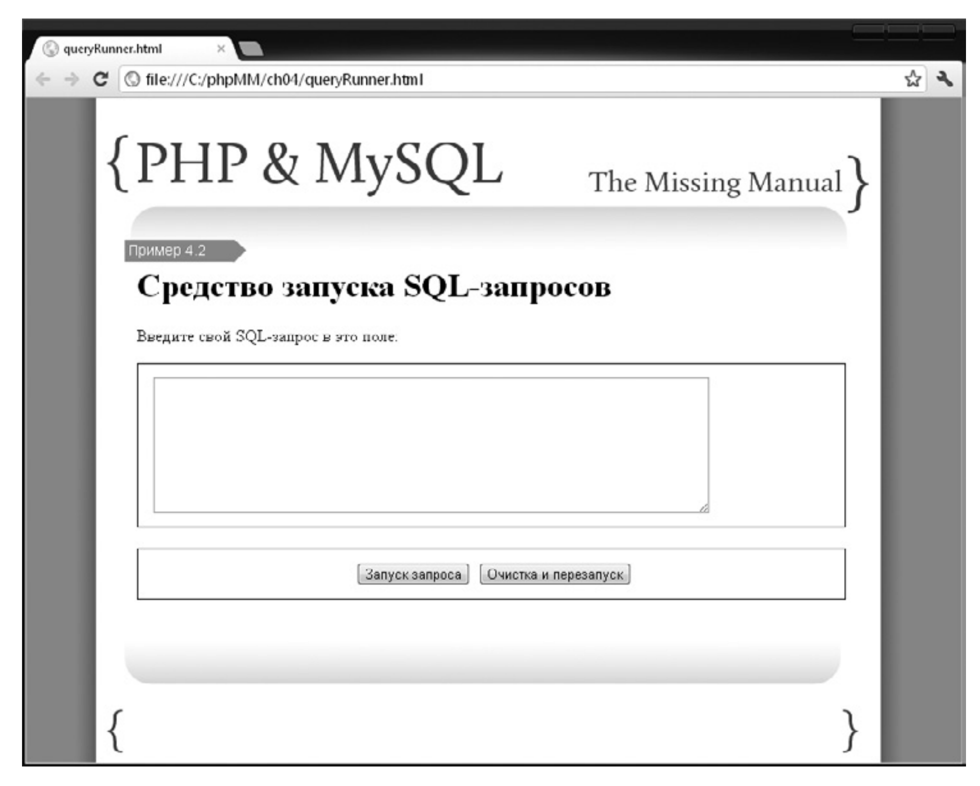


Рис. 4.7. Запуск файла queryRunner.html

Подключение к базе данных (еще раз)

Вы уже знаете, что мы сейчас сделаем: нужно подключиться к MySQL и задействовать нужную базу данных. Этот код уже должен быть вам достаточно знаком. Откройте run_query.php и приступайте к работе:

```
<?php
  require '../../scripts/app_config.php';

  mysql_connect(DATABASE_HOST, DATABASE_USERNAME, DATABASE_PASSWORD)
    or die("<p>Ошибка подключения к базе данных: " .
           mysql_error() . "</p>");

  echo "<p>Вы подключились к MySQL!</p>";
  mysql_select_db(DATABASE_NAME)
    or die("<p>Ошибка при выборе базы данных " . DATABASE_NAME .
           mysql_error() . "</p>");

  echo "<p>Вы подключены к MySQL с использованием базы данных " .
           DATABASE_NAME . "</p>";
?>
```

Но постойте, все это уже знакомо. Этот код вы уже набирали в разделе «Выбор используемой базы данных» данной главы. Получается, что его нужно набирать всякий раз при подключении к MySQL? Такая разновидность дублирования кода вам ни к чему. Именно поэтому мы переместили наши константы базы данных в файл app_config.php: нам нужна возможность держать один и тот же код в одном месте, а не в десятках и сотнях мест.

Вы уже видели, насколько просто запросить файл с помощью инструкции require (в подразделе «Абстрагирование важных значений путем заключения их в отдельный файл» данного раздела) и воспользоваться значениями констант. То же самое можно сделать и с кодом подключения к базе данных. Откройте новый файл и назовите его database_connection.php. Сохраните этот новый сценарий там же, где размещается файл app_config.php, и наберите следующий код:

```
<?php
  require 'app_config.php';

  mysql_connect(DATABASE_HOST, DATABASE_USERNAME, DATABASE_PASSWORD)
    or die("<p>Ошибка подключения к базе данных: " .
           mysql_error() . "</p>");

  echo "<p>Вы подключились к MySQL!</p>";

  mysql_select_db(DATABASE_NAME)
    or die("<p>Ошибка при выборе базы данных " .
           DATABASE_NAME . mysql_error() . "</p>");

  echo "<p>Вы подключены к MySQL с использованием базы данных " .
       DATABASE_NAME . ".</p>";
?>
```

ПРИМЕЧАНИЕ

Убедитесь в том, что путь к вашему файлу app_config.php соответствует месту, где вы сохранили этот файл. Если вы сохраняете database_connection.php в том же каталоге, что и app_config.php, вам понадобится только имя файла без указания путевых имен каталогов.

Теперь весь код подключения к базе данных благополучно перенесен в другое место, следовательно, вы можете полностью изменить содержимое файла run_query.php:

```
<?php
  require '../../scripts/database_connection.php';
?>
```

Ну, как вам нравится такой короткий код? Заметьте еще, что больше не нужно требовать загрузки файла app_config.php. Сценарию требуется файл database_connection.php. И этот тот самый database_connection.php, который в свою очередь привлекает к работе сценарий app_config.php. Теперь ваш код выглядит намного красивее и лаконичнее.

Чтобы убедиться в его работоспособности, нужно зайти на страницу queryRunner.html и щелкнуть на кнопке **Запуск запроса**. Должно появиться изображение, показанное на рис. 4.8. И это при отсутствии всего, кроме одной-единственной инструкции require в вашем основном сценарии!

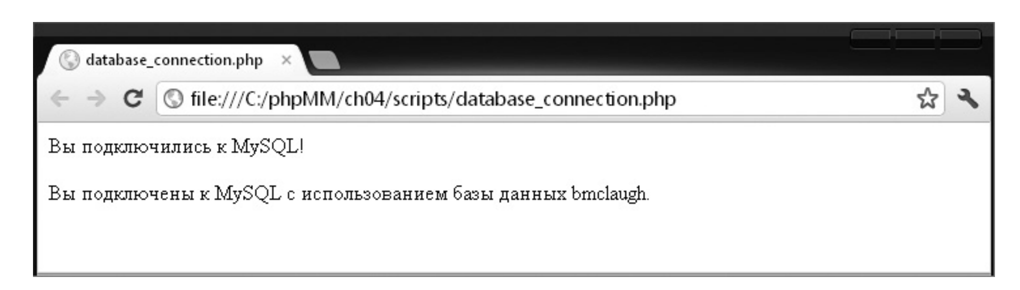


Рис. 4.8. Подключение к базе данных с помощью инструкции require

Создание сценария, который (по крайней мере пока) ничего не делает, кроме использования другого сценария, может показаться странным. Вообще-то, чем больше вы будете разбираться в программировании, тем больше вам будет нравиться такое вот повторное использование. Для выполнения задания достаточно написать новый код. Если вы можете повторно применять 100 или 1000 строк существующего кода, то вы именно это и должны сделать.

Запуск пользовательского SQL-запроса (еще раз)

Здесь вы наконец-то добрались до объединения всех своих знаний о PHP со всем, что вы знаете о SQL. Благодаря переменной $_REQUEST, которая, как вы помните, является массивом (см. раздел «Переменная $_REQUEST» главы 2), у вас уже есть

все, что пользователь поместил в большое многострочное поле ввода вашей формы. Кроме того, для запуска запроса вы можете воспользоваться функцией mysql_query.

Нужно только лишь все это объединить:

```
<?php
  require '../../scripts/database_connection.php';

  $query_text = $_REQUEST['query'];
  $result = mysql_query($query_text);

  if (!$result) {
    die("<p>Ошибка при выполнении SQL-запроса" . $query_text . ": " .
        mysql_error() . "</p>");
  }

  echo "<p>Результаты вашего запроса:</p>";
  echo "<ul>";

  while ($row = mysql_fetch_row($result)) {
    echo "<li>{$row[0]}</li>";
  }
  echo "</ul>";
?>
```

Нужно просто забрать соответствующее поле из ввода вашей HTML-формы, передать его функции mysql_query — и дело в шляпе. Затем можно будет передать возвращенный PHP-ресурс — $result — инструкции if, обрабатывающей ошибку, и наконец функции mysql_fetch_row для вывода результатов запроса на экран.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Почему бы не абстрагировать функцию mysql_query?

Вы могли заметить, что, поскольку вы постоянно подключаетесь к MySQL, всякий раз используя одно и то же имя пользователя и пароль, и снова выбираете базу данных (зачастую одну и ту же), вы также снова и снова будете вызывать функцию mysql_query. По этой причине почему бы не поместить ее в другой файл, которым потом можно будет воспользоваться с помощью инструкции require?

Причина кроется в самом коде предыдущего примера: то, что передается функции mysql_query, склонно к изменениям практически при каждом вызове этой функции. Например, в файле connect.php (см. подраздел «Подключение к базе данных MySQL» раздела «Создание простого PHP-сценария, предназначенного для подключения» данной главы) этой функции передается запрос SHOW TABLES, а в коде предыдущего примера ей отправляется запрос из поля формы в queryRunner.html. Даже при том, что вы вызываете mysql_query снова и снова, передаваемый этой функции аргумент изменяется, поэтому, если убрать данную функцию из основного сценария, это вряд ли вам чем-то поможет.

> Вы можете переместить `mysql_query` за пределы вашего основного сценария и передать ей изменяемую часть инструкции — SQL-запрос. Но вам потребуется создать собственную функцию, забирающую ваш запрос из основного сценария и передающую этот запрос функции `mysql_query`. Затем, когда `mysql_query` завершит свою работу, ваша собственная функция должна будет передать все возвращенное функцией `mysql_query` назад основному сценарию.
>
> Все это может звучать как-то невнятно и потребует большого объема работы. Хотя все это сделать нетрудно, и когда вы начнете создавать собственные функции (а эта работа предстоит вам при изучении главы 8), проблем с решением этой задачи у вас не будет. Но что вы этим выгадаете? Придется по-прежнему передавать запрос и возвращать ответ. От создания своей функции вы ничего не выиграете, она будет в основном подменять собой функцию `mysql_query`, но вы при этом не получите никаких дополнительных функциональных возможностей и она не добавит вашему коду никакой защиты от изменений или чего-либо в этом роде.
>
> Теперь, до того как вы станете размышлять о том, что не стоит беспокоиться о таких вещах, потратьте еще одну минуту. Подумайте вот о чем. Можете ли вы поместить этот код в еще один общий файл? Можете ли вы превратить его в пользовательскую функцию? Это очень прибыльное дело! Вам нужно думать в этом направлении, даже если вы решите, как в данном случае, что этого делать не стоит. Чем больше вы будете обдумывать новые идеи и способы подхода к вашему коду, тем выше будет становиться ваше мастерство программиста. Поэтому постоянно задавайте себе такие вопросы и не бойтесь отвечать на них следующим образом: «Нет, для этого случая данная идея не подходит».

Сценарий выглядит довольно привлекательно, и теперь можно испытать его в деле.

Ввод вашего первого запроса, основанного на применении веб-технологий

На данный момент в вашей базе данных, наверное, не так уж много информации, поэтому начнем с создания новой таблицы под названием `urls`. Вот необходимый для этого код SQL:

```
CREATE TABLE urls (id int, url varchar(100), description varchar(100));
```

Разумеется, при наличии такого шикарного многострочного текстового поля вы можете ввести запрос в развернутой форме:

```
CREATE TABLE urls (
  id int,
  url varchar(100),
  description varchar(100)
)
```

В любом случае вам нужна форма (рис. 4.9). Использование здесь многострочного текстового поля позволяет вашим пользователям вводить SQL код так, как

им это заблагорассудится. Вроде бы мелочь, но эта незначительная гибкость и рассчитанная на предоставление удобств пользователю конструкция доставляет дополнительное удовольствие пользования вашей формой. Вам ведь не захочется записывать длинные SQL-инструкции в гигантское однострочное поле ввода, так зачем же предлагать это вашим пользователям?

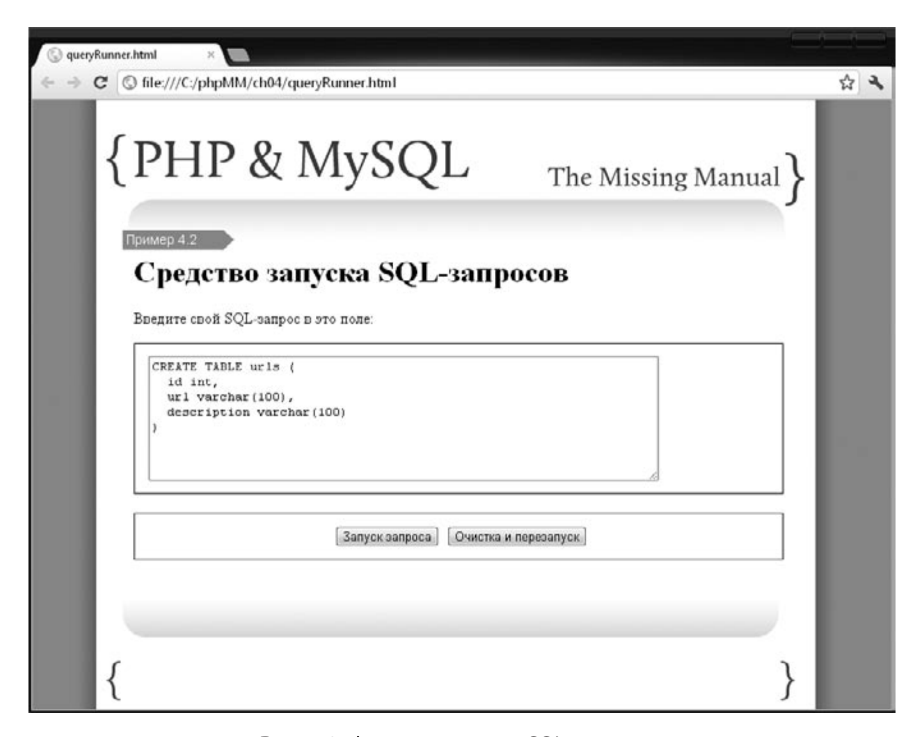


Рис. 4.9. Форма для ввода SQL-запросов

Теперь щелкните на кнопке Запуск запроса. Что из этого вышло? Скорее всего, вы увидите нечто следующее (рис. 4.10). Да, иногда отсутствие сообщения об ошибке хуже самого сообщения. Здесь ничто не поможет разобраться с тем, что пошло не так в сценарии. В подобных случаях поможет не вызывающее раздражения сообщение об ошибке.



Рис. 4.10. Что-то пошло не так

Итак, что же получилось? Если вы хотите окончательно запутаться, щелкните на кнопке возврата к предыдущей странице браузера и запустите еще раз свой запрос CREATE. Вы увидите сообщение, подобное показанному на рис. 4.11.



Рис. 4.11. Сообщение об ошибке

При первом запуске запроса CREATE TABLE вы не получили в ответ вообще никакого результата. Во второй раз программа MySQL сообщает вам, что таблица urls уже существует! И действительно, если перейти в окно командной строки, вы увидите, что таблица уже присутствует в вашей базе данных:

```
mysql> describe urls;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id          | int(11)      | YES  |     | NULL    |       |
| url         | varchar(100) | YES  |     | NULL    |       |
| description | varchar(100) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Еще раз внимательно изучите свой код:

```
<?php
  require '../../scripts/database_connection.php';

  $query_text = $_REQUEST['query'];
  $result = mysql_query($query_text);

  if (!$result) {
    die("<p>Ошибка при выполнении SQL-запроса " . $query_text . ": " .
        mysql_error() . "</p>");
  }

  echo "<p>Результаты вашего запроса:</p>";
  echo "<ul>";
  while ($row = mysql_fetch_row($result)) {
    echo "<li>{$row[0]}</li>";
  }
  echo "</ul>";
?>
```

Блок кода `if (!$result)` не выполнялся, поэтому вполне понятно, что результат в переменной `$result` вернулся не со значением `false`. Но цикл `while` так и не запустился, поскольку вы так и не увидели результатов.

Но постойте, вы отправляли запрос на создание — `CREATE`. Какие же тогда строки будут возвращены запросом такого типа? О каких строках вообще может идти речь, когда никакие строки не запрашивались? Вы просто попросили MySQL создать таблицу в качестве места, куда помещаются строки.

Обработка запросов, не выбирающих информацию с помощью команды SELECT

У функции `mysql_query` есть одна особенность: она без возражений принимает инструкцию `CREATE`. Она даже делает то, что от нее требуют, именно поэтому при повторном введении этого же запроса MySQL выдает ошибку, сообщая о том, что таблица `urls` уже была создана. Когда функция `mysql_query` получает инструкцию `CREATE`, она возвращает `false` в случае возникновения ошибки, которую обрабатывает ваш сценарий, и возвращает `true`, если ошибки не было. И если ошибка не возникла, эта функция не возвращает никаких строк. Вы получаете в переменной `$result` только значение `true` и ничего больше. Именно здесь события развиваются неправильно.

Когда функция `mysql_query` получает большинство SQL-инструкций, не занимающихся выборкой данных, среди которых `CREATE`, `INSERT`, `UPDATE`, `DELETE`, `DROP` и некоторые другие инструкции, она просто возвращает значение `true` (если все получается) или `false` (если выполнить инструкцию не удается).

ПРИМЕЧАНИЕ

Некоторые из этих SQL-инструкций, например UPDATE и DELETE, могут быть вам незнакомы. Но волноваться не стоит. Их предназначение обуславливается их именами: UPDATE обновляет информацию в таблице, а DELETE ее удаляет. И когда понадобится воспользоваться этими функциями в данной книге, вы узнаете дополнительные подробности о конкретном способе применения каждой из них.

Теперь, когда вы уже знаете, что происходит, будет несложно справиться с данной проблемой. Нужно, чтобы ваш сценарий определял наличие в строке SQL-запроса, предоставленного пользователем этих специальных слов. В таком случае обработка должна отличаться от обычной. А с поиском внутри строк вы уже достаточно знакомы.

Итак, обдумаем задачу. Нужно сделать следующее.

1. Забрать пользовательский запрос из HTML-формы.
2. Передать запрос функции `mysql_query` и сохранить результат в переменной.
3. Определить, не имеет ли результат значение `false`, свидетельствующее о неудачном исходе SQL-запроса любого типа.

4. Если значение результата не равно `false`, определить, не имелось ли в запросе одно из специальных ключевых слов: `CREATE`, `INSERT`, `UPDATE`, `DELETE` или `DROP`. (Есть и другие слова для проверки, но здесь присутствуют только самые распространенные.)
5. Если в запросе имеется одно из этих специальных слов, определить, было ли значение результата равно `true`, и оповестить пользователя об успешном выполнении запроса.
6. Если в запросе одного из этих слов нет, вывести строки результата, как это уже делалось.

У вас уже есть основная часть этого кода, поэтому вам нужно лишь добавить `if` (или, может быть, несколько инструкций `if`) и поиск в строке:

```
$return_rows = false;
$location = strpos($query_text, "CREATE");
if ($location === false) {
  $location = strpos($query_text, "INSERT");
  if ($location === false) {
    $location = strpos($query_text, "UPDATE");
    if ($location === false) {
      $location = strpos($query_text, "DELETE");
      if ($location === false) {
        $location = strpos($query_text, "DROP");
        if ($location === false) {
          // Если выполнение кода добралось до этого места,
          // значит этот запрос не CREATE, INSERT, UPDATE, DELETE
          // или DROP. Он должен вернуть строки.
          $return_rows = true;
        }
      }
    }
  }
}
```

ВНИМАНИЕ

В инструкциях if для проверки значения $location на false обязательно должен использоваться тройной знак равенства (= = =).

Этот код может показаться сложным, но в нем нетрудно разобраться, если просмотреть его построчно. По сути, здесь имеется одна и та же инструкция `if`, которая повторяется снова и снова, и у каждой из инструкций есть другая, вложенная в нее инструкция `if`:

```
$location = strpos($query_text, "SEARCH_STRING");
if ($location === false) {
  // Новая попытка с другой SEARCH_STRING
}
```

И наконец, если все инструкции if не сработают, значит, в строке запроса отсутствует слово CREATE, INSERT, UPDATE, DELETE или DROP:

```
// Это самая внутренняя инструкция if
if ($location === false) {
  // Если выполнение кода добралось до этого места, значит,
  // этот запрос не CREATE, INSERT, UPDATE, DELETE или DROP.
  // Он должен вернуть строки.
  $return_rows = true;
}
```

Но к чему такие сложности? Проблема в том, что в действительности нужно провести поиск в строке пользовательского запроса не только одного соответствующего слова, такого как CREATE или INSERT, но нескольких соответствующих строк. Это непросто, поэтому всякий раз приходится работать с одним вызовом функции strpos.

ПРИМЕЧАНИЕ

Вам нужно разобраться с работой этого кода, но не брать его за основу. Он далеко не идеален. В следующей главе в ваш PHP-набор программиста будет добавлен совершенно новый инструмент. Этот код будет переработан, причем станет намного лаконичнее и элегантнее.

На каждом этапе если искомая строка будет найдена, значит пользователь вставил одно из этих специальных ключевых слов SQL, которое не возвращает строки, поэтому переменная $return_rows имеет значение false, отличающееся от его исходного значения true.

И в конце всего этого парада фигурных скобок инструкции if возвращаются назад к основной программе и переменная $returns_rows имеет либо значение true, если ни один из поисков не дал совпадений, либо false, если одно из совпадений все же состоялось.

Теперь вы готовы использовать $returns_rows для вывода результата:

```
<?php
  // код инструкций require и подключения к базе данных

  // запуск запроса

  // обработка ошибок в результате

  // определение, является результат запроса набором строк или нет

  if ($return_rows) {
    // имеются строки для показа в качестве результата запроса
    echo "<p>Результаты вашего запроса:</p>";
    echo "<ul>";
    while ($row = mysql_fetch_row($result)) {
      echo "<li>{$row[0]}</li>";
    }
    echo "</ul>";
  } else {
```

```
    // Строки отсутствуют. Вывод простого отчета о том,
    // работал запрос или нет.
    if ($result) {
      echo "<p>Следующий запрос был обработан успешно:</p>"
      echo "<p>{$query_text}</p>";
    }
  }
?>
```

ПРИМЕЧАНИЕ

Следует вспомнить, что if ($return_rows) — это то же самое, что и if ($return_rows = = = true). То же самое относится и к if ($result).

Основная часть этого сценария вам уже знакома. Весь код, использовавшийся для вывода строк, остался прежним. Данный код просто перемещается в блок `if ($return_rows)`, поскольку он применяется, только если пользователь поместил в поле что-нибудь вроде инструкции `SELECT`, которая возвращает (потенциально) множество результатов.

Затем в блоке `else` этой инструкции `if` сценарий просто выводит отчет о том, все ли прошло удачно. В качестве дополнительной справки этот блок инструкции `if` выводит текст исходного запроса, чтобы пользователь знал, что было выполнено.

Теперь чисто технически вам в действительности не нужна эта инструкция `if ($result)`. Поскольку ранее уже было проведено тестирование значения переменной `$result` на `false`. Если ваш сценарий добрался до этого последнего фрагмента сценария, вы знаете, что `$result` имеет значение `true`, поэтому последний фрагмент можно упростить:

```
if ($return_rows) {
  // запрос вернул строки для показа
  echo "<p>Результаты вашего запроса:</p>";
  echo "<ul>";
  while ($row = mysql_fetch_row($result)) {
    echo "<li>{$row[0]}</li>";
  }
  echo "</ul>";
} else {
  // Строки отсутствуют. Вывод простого отчета о том,
  // работал запрос или нет.
  echo "<p>Следующий запрос был обработан успешно:</p>"
  echo "<p>{$query_text}</p>";
}
```

Этот сценарий получился немного длиннее, но теперь вы уже знаете, для чего предназначена каждая его строка. Испытаем его в работе.

Вполне вероятно, что у вас уже создана таблица `urls`, даже если ваш PHP-сценарий не оповестил вас об этом. Поэтому попробуйте в качестве SQL-запроса ввести `DROP TABLE urls;`. Затем запустите запрос, на этот раз вы должны получить

полезное сообщение, которое характерно для вашего запроса, не возвращающего строки (рис. 4.12).

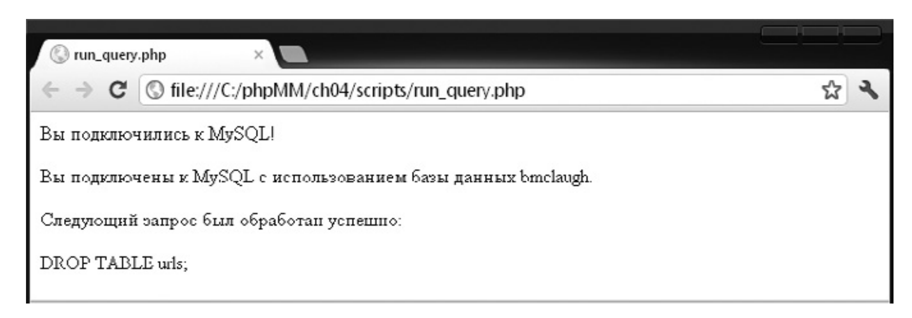


Рис. 4.12. Ответ на запрос

Теперь последняя версия сценария `run_query.php` определяет, что ей был передан запрос, содержащий одно из ключевых слов, которые свидетельствуют об отсутствии возвращенных строк. Сообщение при успешном выполнении запроса все еще не слишком многословное, но по крайней мере это уже не пустое место, которое выводилось кодом, пытающимся вывести отсутствующие в данном случае строки ответа на запрос.

Учет человеческого фактора

К сожалению, в одной из строк кода, показанного в предыдущем разделе, по-прежнему кроется нерешенная проблема. Что если именно сейчас ваш пользователь введет следующий запрос:

```
DROP TABLE urls;
```

Ваш набор инструкций обнаружит, что слово `DROP` является частью запроса, придет к выводу, что возвращенных строк не будет, и выполнит предписанное: выведет отчет либо об успешном выполнении запроса, либо о возникновении ошибки.

А как насчет следующего запроса? Сможете ли вы понять, в чем здесь проблема:

```
drop table urls;
```

Вот как выглядит инструкция, которая должна определить соответствие:

```
$location = strpos($query_text, "DROP");
if ($location === false) {
  // Инструкция должна вернуть true, показав,
  // что возвращаемые строки отсутствуют.
}
```

Но в этой строке ведется поиск слова `DROP`, которое не соответствует слову drop. Функция `strpos` ищет в строках, но она считает, что буква нижнего регистра, например, d отличается от буквы верхнего регистра D. Поэтому в данной строке ведется поиск слова DROP, а не drop или dRoP.

Ну и кроме того, вашим приложением пользуются люди, а не роботы. Вы можете, конечно, предположить, что эти люди хорошо разбираются в SQL и всегда применяют буквы верхнего регистра. Вы можете поместить в форму небольшое сообщение: «Пожалуйста, вводите код SQL заглавными буквами», но люди есть люди, и они склонны игнорировать подобные инструкции.

Учет в коде человеческого фактора займет чуть ли не столько же времени, сколько написание кода, обрабатывающего так называемый нормальный ход событий. Ну раз уж вы ориентированы на работу с реальными людьми, нужно уяснить, что понятие «нормальный» здесь употреблять бесполезно. Вместо этого ваш код должен справляться со всеми мыслимыми (и даже немыслимыми) сценариями развития ситуации.

Устранить проблему больших и малых букв нетрудно: нужно просто до начала поиска в значении переменной `$query_string` перевести в верхний регистр все имеющиеся буквы:

```
$return_rows = false;
$query_text = strtoupper($query_text);
$location = strpos($query_text, "CREATE");
// все вложенные блоки if
```

Теперь, если пользователь наберет `drop table urls` или `DROP table UrLS`, строкой для поиска будет `DROP TABLE URLS` и поиск слова DROP вернет соответствие.

Но есть еще одна проблема! Перед тем как прочитать о ней, попробуйте обнаружить ее самостоятельно.

ПРИМЕЧАНИЕ

Да, даже в одной небольшой программе может встретиться так много шероховатостей и проблем. Именно поэтому самих программистов очень много, но по-настоящему великих среди них довольно мало. Различие между ними состоит в умении справляться со всеми этими мелкими деталями, не швыряя свой iPhone в ближайшую стену.

По возможности нужно избегать внесения изменений в пользовательский ввод

Чтобы заметить еще одну потенциальную проблему в своем сценарии запросов, посмотрите на последний фрагмент кода, который запускается, если пользователь ввел запрос, не возвращающий строки, например `DROP` или `INSERT`:

```
// Строки отсутствуют. Вывод простого отчета о том, работал запрос или нет.
echo "<p>Следующий запрос был обработан успешно:</p>"
echo "<p>{$query_text}</p>";
```

Запустите этот код, и вы увидите картинку, похожую на рис. 4.13.

ПРИМЕЧАНИЕ

Если вы практически выполняете все, что здесь рассматривается, то перед повторным созданием таблицы с помощью команды CREATE может понадобиться воспользоваться командой DROP TABLE urls;. Или же до того, как вы сможете удалить таблицу с помощью команды DROP, может понадобиться создать ее с помощью команды CREATE.

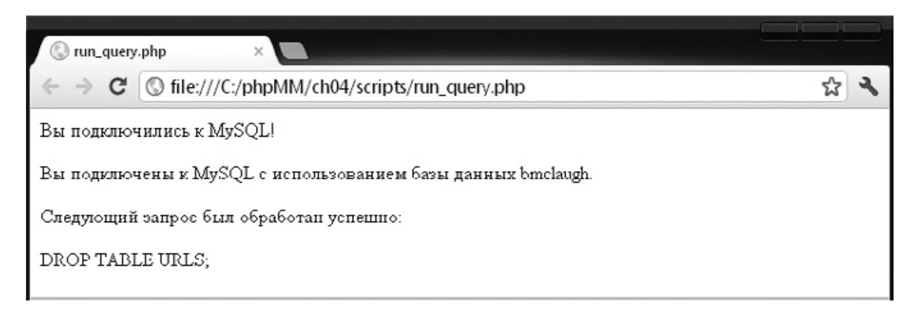


Рис. 4.13. Исправленный SQL-запрос (сравните с запросом на рис. 4.12)

Внимательно сравните код на рис. 4.12 и на рис. 4.13, и вы увидите, что запрос целиком выведен заглавными буквами. И понятно почему. Чтобы облегчить поиск, вы добавили к вашему сценарию следующую строку:

```
$query_text = strtoupper($query_text);
```

Затем в конце сценария значение переменной `$query_text` выводится заглавными буквами. Имеет ли это какое-то значение? Возможно, и нет, по крайней мере здесь. Но это кое о чем все же говорит: после приведения всех букв переменной `$query_text` к верхнему регистру ее значение возвращается именно в таком виде.

Предположим, что исходный запрос имел следующий вид:

```
SELECT *
  FROM users
 WHERE last_name = "MacLachlan";
```

Теперь рассмотрим тот же самый запрос, все буквы которого приведены к верхнему регистру:

```
SELECT *
  FROM USERS
 WHERE LAST_NAME = "MACLACHLAN";
```

И это уже совсем не тот запрос. Инструкция `SELECT`, как и большинство других SQL-запросов, будет рассматривать фамилию MacLachlan как совершенно непохожую на MACLACHLAN. Поэтому эти два запроса вообще не идентичны.

В данном случае это не создает проблем. Сценарий никогда не перезапускает один и тот же запрос, и функция `mysql_query` запускается с аргументом `$query_text` еще до перевода его в версию с буквами верхнего регистра. Но эта проблема ждет своего часа.

Вообще-то следует избегать непосредственного изменения ввода, полученного от пользователя, поскольку вы столкнетесь именно с такой разновидностью проблемы: этот ввод может потребоваться еще раз, а так как он изменен, вы не можете вернуть все назад.

К счастью, проблема решается очень просто: вместо изменения пользовательского ввода нужно для хранения версии запроса с буквами в верхнем регистре воспользоваться новой переменной:

```
$return_rows = false;
$uppercase_query_text = strtoupper($query_text);
$location = strpos($uppercase_query_text, "CREATE");
```

Эту новую переменную необходимо использовать во всех поисках в строках:

```
$return_rows = false;
$uppercase_query_text = strtoupper($query_text);
$location = strpos($uppercase_query_text, "CREATE");
if ($location === false) {
  $location = strpos($uppercase_query_text, "INSERT");
  if ($location === false) {
    $location = strpos($uppercase_query_text, "UPDATE");
    if ($location === false) {
      $location = strpos($uppercase_query_text, "DELETE");
      if ($location === false) {
        $location = strpos($uppercase_query_text, "DROP");
        if ($location === false) {
          // Если выполнение кода добралось до этого места,
          // значит, этот запрос не CREATE, INSERT, UPDATE,
          // DELETE или DROP. Он должен вернуть строки.
          $return_rows = true;
        }
      }
    }
  }
}
```

Каким бы незначительным ни было это изменение, оно защищает вас в случае необходимости повторного использования строки запроса.

Теперь, похоже, вы получили средство, которое выполняет любой помещенный в него SQL-запрос. Но оно все еще требует некоторых доработок. Весь этот код поиска в строке загромождает ваш сценарий, и пока эта проблема не решена: на первый, да и на второй взгляд, в сценарии весьма непросто разобраться. В следующей главе мы сделаем из этой элементарной формы по-настоящему красивое упражнение по раскрытию потенциала PHP.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Учет особенностей позиции и удаление пробельных символов

В этой главе вы, конечно же, существенно улучшили сценарий `run_query.php`, но некоторые проблемы все же остались нерешенными. Предположим, кто-то ввел запрос следующего вида:

```
SELECT *
  FROM registrar_activities
 WHERE name = 'Update GPA'
    OR name = 'Drop a class'
```

Здесь используется инструкция SELECT, следовательно, run_query.php должен запустить SQL и вывести все строки, возвращенные этим запросом. Но здесь есть одна небольшая проблема.

Код, ведущий поиск слов update и drop, сообщит о том, что в этом запросе имеются оба этих слова, и просто вернет сообщение, что запрос был выполнен без всяких проблем. Но в этом-то и заключается проблема!

Для ее решения нужно вспомнить структуру SQL. Все эти ключевые слова — CREATE, INSERT и сотоварищи — всегда стоят в запросе первым словом. Следовательно, вам нужно получить позицию соответствия и убедиться, что она равна нулю. Это можно сделать путем дополнения условия if и использования в PHP оператора or (или):

```
if ($location = = = false ||
    $location > 0) {
```

Двойная вертикальная черта означает в PHP «или». Следовательно, в строке кода говорится: «Если вообще нет совпадения ($location = = = false) или совпадение не начинается с первой позиции (с позиции 0), то перейти к поиску следующего ключевого слова». Разумеется, вам придется внести изменения во все инструкции if, что вряд ли доставит удовольствие. Итак, это исправление улучшает код, но с одной реальной помехой еще придется справиться.

Но может сложиться еще более неприятная ситуация! Вы имеете дело с людьми, а им свойственны странности. Предположим, что кто-то ввел в форму следующий код SQL:

```
 CREATE TABLE urls (
     id int,
     url varch*ar(100),
     description varchar(100)
 );
```

Теперь у вас возникла новая проблема: это запрос не на выборку (в нем не используется инструкция SELECT), но ваш код поиска не обнаружит в начале запроса одно из специальных слов. Первым символом здесь стоит простой пробел: " ".

Но эту проблему также можно решить, используя еще одну уже знакомую функцию: trim. Она избавляет строку от окружающих ее пробельных символов, и если ее применить перед проведением поиска, то ситуация улучшится:

```
$uppercase_query_text =
        trim(strtoupper($query_text));
```

Наверное, вам показалось, что вокруг простейшей формы, содержащей всего лишь одно многострочное поле, был проделан слишком большой объем работы. Но при работе с пользовательским вводом нужно постоянно думать, что могут сделать пользователи. И как вы можете уберечь их от созерцания какой-нибудь странной картинки или от внесения ошибок. Думайте в этом направлении, и вы станете создавать более качественные, стабильно работающие и более приятные веб-приложения.

5 Улучшение поиска с помощью регулярных выражений

В предыдущей главе вы сделали то, чем чаще всего занимаются программисты: написали код, решающий задачу. Но он оказался некрасив, неаккуратен и труден для понимания. К сожалению, большинство программистов *оставляют* свой код именно в таком состоянии, потому что... ну он же работает.

Плохой код сродни протекающему крану или неудачно спроектированному каркасу здания. Со временем недостатки проявятся, и кому-то придется решать проблемы. И если вам когда-либо приходилось сталкиваться с тем, что электрик требует с вас денег за то, что его *предшественник* все сделал неправильно, то вы знаете, насколько дорого может обходиться исправление чужих ошибок.

Однако даже хороший код иногда может отказать. Когда вы имеете дело с системой с участием человека, кто-нибудь со временем выдаст что-нибудь неожиданное, то, с чем вы никак не предполагали столкнуться при написании кода. А теперь представьте, что *вы и есть* тот самый электрик, пытающийся исправить для недовольного клиента то, в чем некого обвинить.

Поэтому написание уродливого, но работоспособного кода не для нас. А код в сценарии `run_query.php` в его нынешнем виде оставляет желать лучшего. Как вам все эти инструкции `if`, с помощью которых делается попытка определить, не ввел ли пользователь `CREATE`, `UPDATE`, `INSERT`, `SELECT`... или кто его знает что еще? Ведь в действительности вам нужен способ провести поиск в поступившем запросе всех этих ключевых слов за один раз. А когда все буквы переводятся в верхний регистр, удаляются лишние пробелы и проверяется, что нужное ключевое слово находится в самом начале запроса, все, несомненно, усложняется.

К сожалению, с помощью ранее проделываемых манипуляций со строками и использованием функции `strpos` элегантного решения этой задачи не существует. Но есть другой вариант: *регулярные выражения*. Они похожи на большую бочку с порохом: обладают исключительной эффективностью, но запросто могут взорвать программу и привести к длительному нервному расстройству. Безопасного способа овладеть их мощью не существует.

Но пока все в порядке и ввязываться в бой не нужно. Перед тем как заняться сценарием `run_query.php`, вы получите представление о регулярных выражениях

и сократите все эти раздражающие инструкции if, связанные с поисками в значении переменной $query_text, до одной инструкции. Самое главное — ваша программа станет понятнее, а значит, будет проще выявить причины возникновения проблем в процессе ее работы.

ВНИМАНИЕ

Среди большинства программистов бытует мнение, что регулярные выражения слишком сложны, в них трудно разобраться и вообще они являются неким тайным искусством программирования. На самом деле все не так, и вы абсолютно готовы к освоению регулярных выражений. Как только вы поймете, как они работают, вы будете удивлены, почему кому-то не хочется использовать их везде, где только можно.

Сопоставление строк, двойная скорость

До сих пор для поиска в строках использовалась функция strpos, которой передавались строка, а затем несколько дополнительных символов или строк, которые нужно было искать. Проблема в том, что такой способ применения strpos ограничивал вас одним поиском в строке за один вызов функции, то есть вы могли искать UPDATE и DROP, но только не одновременно.

И тут настает пора воспользоваться регулярными выражениями. *Регулярное выражение* означает регулярную последовательность символов или чисел или какого-нибудь другого шаблона, то есть выражение, которое нужно найти. Итак, если у вас есть строка вроде "abcdefghijklmnopqrstuvwxyz", то вы можете искать в ней шаблон, или регулярное выражение "abc". Оно будет обнаружено только один раз, поэтому не кажется слишком регулярным.

Но, предположим, что у вас есть целая веб-страница и вы хотите поискать ссылки. Чтобы найти все элементы ссылок, можно воспользоваться выражением "<a". Вы можете вообще ничего не найти или найти одну или десять ссылок. Используя регулярное выражение, вы можете искать практически все, что угодно. Все это действительно кажется немного непонятным, поэтому лучше начать с самого начала.

Простая программа поиска в строке

Самым простым регулярным выражением является одиночная простая буква, например "a" или "m". Регулярное выражение "a" будет соответствовать любой букве "a". Пока все очень просто, не так ли?

В PHP, если нужен поиск с использованием регулярного выражения, используется функция preg_match (preg означает PHP regular (expressions), регулярные выражения). Эта функция используется следующим образом:

```
<?php
$string_to_search = "Martin OMC-28LJ";
$regex = "/OM/";
$num_matches = preg_match($regex, $string_to_search);
```

```
if ($num_matches > 0) {
  echo "Соответствие найдено!";
} else {
  echo "Соответствия отсутствуют. Извините.";
}
?>
```

ВНИМАНИЕ

Первым функции preg_match должно быть передано регулярное выражение, а не строка, в которой нужно вести поиск. Такое расположение может показаться обратным по отношению к тому, с которым вы работали, но вскоре вы станете использовать функцию preg_match и родственные ей функции настолько часто, что передача первой строки, в которой ведется поиск, станет казаться странной.

Приступим к работе. Сохраните эту программу под именем regex.php и запустите ее из командной строки. Результат должен быть следующим:

```
--(08:25 $)-> php regex.php
Соответствие найдено!
```

Надо сказать, что он не слишком впечатляющий. Однако это как в детстве: прежде чем вы сможете ходить, нужно научиться ползать. И одним из шагов на данном этапе станет понимание того, как нужно составлять регулярное выражение.

Начнем с того, что регулярное выражение — простая строка, поэтому его нужно взять в кавычки. Обычно будут использоваться двойные ("), а не одинарные кавычки ('), потому что вам придется применять некоторые необычные escape-символы, а PHP не выполняет такую полезную обработку внутри строк в одинарных кавычках, но делает это внутри строк в двойных кавычках.

Кроме того, регулярные выражения начинаются и заканчиваются с прямого слэша. Все, что находится между этими слэшами, и составляет суть выражения. Следовательно, "/OM/" является регулярным выражением, ведущим поиск строки OM.

Если говорить точнее, "/OM/" ищет исключительно OM. Оно не будет соответствовать om, Om или OhM. Для него нужно, чтобы за прописной буквой O следовала прописная буква M. Иными словами, пока это похоже на соответствие строке, с которым вы уже работали.

У функции preg_match также есть свои особенности. Как вы уже видели, сначала ей передается регулярное выражение, а затем строка, в которой нужно вести поиск. Затем она возвращает количество соответствий, а не позицию, в которой было найдено соответствие. А вот вам и первая настоящая неприятность: функция preg_match никогда не вернет ничего, кроме 0 или 1. Она возвращает 0 при отсутствии соответствий, 1 — при обнаружении первого соответствия, а затем просто останавливает поиск.

Если нужно найти все соответствия, следует воспользоваться функцией preg_match_all. Поэтому выражение preg_match("/Mr/", "Mr. Mranity") возвращает 1, а выражение preg_match_all("/Mr/", "Mr. Mranity") возвращает 2.

ПРИМЕЧАНИЕ

Есть еще аргументы, которые могут быть переданы функциям preg_match и preg_match_all, а также результаты, которые могут быть получены от них. Обо всех этих возможностях можно узнать в интерактивном режиме, зайдя на сайт www.php.net/manual/en/function.preg-match.php.

Поиск одной строки... или другой

Пока создается впечатление, что регулярные выражения не предлагают чего-либо такого, что нельзя было получить при использовании функции strpos. Но их возможности гораздо шире, и одной из самых впечатляющих является поиск одной или другой строки. Для него используется специальный символ, называемый каналом. Он выглядит как вертикальная черта: |.

ПОД КАПОТОМ

Какая из кавычек лучше?

Практически в каждом языке программирования строки в одинарных кавычках (‘My name is Bob’) и строки в двойных кавычках (“I am a carpenter”) рассматриваются одинаково. Но также практически в каждом языке программирования последствия применения тех или иных кавычек заходят гораздо дальше того, что вы можете себе представить.

В целом в одинарных кавычках ведется *меньше* обработки. Но какая обработка ведется в первую очередь? Возьмем утверждение I’m going to the bank. Если поместить его в одинарные кавычки, получится ‘I’m going to the bank’. Но PHP станет на вас ругаться, потому что одинарная кавычка в I’m похожа на завершение простой строки ‘I’, а все остальное — m going to the bank — должно быть чем-то другим. Разумеется, это не то, что вы имели в виду, поэтому вы выполняете одно из двух действий: либо переходите на использование двойных кавычек и продолжаете работать дальше, либо нейтрализуете одинарную кавычку.

Нейтрализация какого-нибудь символа означает сообщение языку программирования о том, что не нужно рассматривать что-то как часть языка и что это что-то является всего лишь частью строки. Обычно символы нейтрализуются путем вставки обратного слэша (\) впереди потенциально проблемного символа.

Поэтому строку I’m going to the bank. в одинарных кавычках можно записать как ‘I\’m going to the bank’. Символ \ заставляет PHP проигнорировать как этот символ, так и тот, который за ним следует.

А что если вам понадобится записать сам обратный слэш? Предположим, что вы пишете программу для своего старенького прадедушки, который все еще работает в MS-DOS на своей 286 машине? Вам может понадобиться сказать ‘Никогда и ни при каких условиях не следует набирать \’del C:*.*\’ и нажимать Enter!’ Итак, вы успешно обработали одинарные кавычки, но теперь PHP пытается нейтрализовать символ, который следует за обратным слэшем, имеющимся внутри строки: *.

Для PHP возникает весьма запутанная ситуация, он не может определить, зачем его заставляют нейтрализовать звездочку. В данном случае вам нужно нейтрализовать сам обратный слэш. Поэтому вы просто вставляете в строку

escape-символ — обратный слэш, — а затем тот символ, который должен быть нейтрализован: еще один обратный слэш. В результате получается 'Никогда и ни при каких условиях не следует набирать \'del C:*.*\' и нажимать Enter!'

За исключением действия с одиночной кавычкой (') и обратным слэшем (\), PHP не ведет никакую другую обработку в ваших строках, заключенных с одинарные кавычки. Но есть еще множество других случаев, когда может понадобиться обработка: символ новой строки (`\n`), табуляции (`\t`) или тот самый ловкий способ вставки значений переменных непосредственно в строку с помощью синтаксиса `{$variable}` или просто использование переменной `$variable`.

Итак, возможности строки в одинарных кавычках весьма ограниченны. Все виды дополнительной обработки доступны только при использовании строки, заключенной в двойные кавычки. В результате большинство программистов стремятся применять двойные кавычки. В этом случае не нужно раздумывать: «Понадобится ли мне дополнительная обработка внутри строки или могу ли я использовать одинарные кавычки?»

И еще одно последнее замечание: дополнительная обработка не создает проблем снижения производительности в 99 % создаваемых вами приложений. Вычисления, необходимые для обработки этих дополнительных escape-символов и переменных, не мешают работе ваших заказчиков и не перегружают серверные жесткие диски или микросхемы оперативной памяти. Вы можете спокойно пользоваться исключительно строками, заключенными в двойные кавычки, и при этом никогда не сталкиваться вообще ни с какими проблемами.

Когда понадобится вести поиск по одному или по другому аргументу, оба эти аргумента помещаются в круглые скобки и разделяются символом вертикальной черты:

```
/(Mr|Dr)\. Smith/
```

Обратите внимание на следующую особенность: наличие обратного слэша (\). Он нейтрализует точку, поскольку в регулярном выражении точка обычно означает «соответствовать любому одиночному символу». А в данном случае нужно именно соответствие точке и не чему-нибудь другому. Поэтому `\.` будет соответствовать точке и ничему, кроме точки.

`/Mr. Smith/` будет соответствовать строке Mr. Smith, а вот строку Dr. Smith будет пропускать. А вот `/(Mr|Dr). Smith/` будет соответствовать строке Mr. Smith или строке Dr. Smith.

Поэтому данный небольшой фрагмент кода будет находить соответствие в обоих случаях:

```
// Здесь будет найдено соответствие
echo "Количество соответствий: " . preg_match("/(Mr|Dr). Smith/", "Mr. Smith");

// И здесь также будет найдено соответствие
echo "Количество соответствий: " . preg_match("/(Mr|Dr). Smith/", "Dr. Smith");
```

Используя это новое качество, можно будет внести весьма существенные изменения в сценарий run_query.php из предыдущей главы. Откройте этот файл и посмотрите на его содержимое. Его старая версия имеет следующий вид:

```
<?php
  require '../../scripts/database_connection.php';

  $query_text = $_REQUEST['query'];
  $result = mysql_query($query_text);

  if (!$result) {
    die("<p> Ошибка при выполнении SQL-запроса: " . $query_text . ": " .
        mysql_error() . "</p>");
  }

  $return_rows = false;
  $uppercase_query_text = strtoupper($query_text);
  $location = strpos($uppercase_query_text, "CREATE");
  if ($location === false) {
    $location = strpos($uppercase_query_text, "INSERT");
    if ($location === false) {
      $location = strpos($uppercase_query_text, "UPDATE");
      if ($location === false) {
        $location = strpos($uppercase_query_text, "DELETE");
        if ($location === false) {
          $location = strpos($uppercase_query_text, "DROP");
          if ($location === false) {
           // Если выполнение кода добралось до этого места, значит
           // этот запрос не CREATE, INSERT, UPDATE, DELETE или DROP.
           // Он должен вернуть строки.
            $return_rows = true;
          }
        }
      }
    }
  }

  if ($return_rows) {
    // Запрос вернул строки для показа
    echo "<p> Результаты вашего запроса:</p>";
    echo "<ul>";
    while ($row = mysql_fetch_row($result)) {
      echo "<li>{$row[0]}</li>";
    }
    echo "</ul>";
  } else {
    // Строки отсутствуют. Вывод простого отчета о том,
    // работал запрос или нет.
    echo "<p>Следующий запрос был обработан успешно:</p>";
    echo "<p>{$query_text}</p>";
  }
?>
```

Такое количество инструкций if портит внешний вид сценария. Но используя регулярные выражения, можно внести весьма эффектные изменения:

```
<?php
  // код инструкций require и подключения к базе данных

  $return_rows = true;
  if (preg_match("/(CREATE|INSERT|UPDATE|DELETE|DROP)/",
                 strtoupper($query_text))) {
    $return_rows = false;
  }

 if ($return_rows) {
    // вывод кода
  }
?>
```

ПРИМЕЧАНИЕ

Чтобы можно было вернуться назад и посмотреть, с чего все начиналось, эту версию нужно сохранить в другом файле или в другом каталоге. В примерах для данной книги исходную версию run_query.php можно найти в каталоге примеров для главы 4, а новую версию — в каталоге примеров для главы 5.

Присмотритесь к этому коду, особенно к довольно длинному условию для инструкции if. Разберем все происходящее поэтапно.

1. Сначала переменной $return_rows присваивается вместо значения false значение true. Причина в том, что с помощью регулярного выражения ведется поиск, определяющий отсутствие возвращаемых строк. Эта версия читается легче предыдущей, в которой делались постоянные сравнения и при отсутствии соответствий значение $return_rows устанавливалось в true.
2. Условие if начинается с функции preg_match. Здесь нет необходимости использовать функцию preg_match_all, поскольку вас интересует вопрос, есть ли вообще искомые строки, а не вопрос, не появляются ли они более одного раза.
3. В коде применяется довольно простое регулярное выражение: в нем содержатся все ключевые слова для SQL-инструкций, не возвращающих строк. Эти ключевые слова разделены знаком вертикальной черты. Таким образом, это выражение для поиска соответствий в строке, содержащей CREATE, *или* INSERT, *или* UPDATE, *или* DELETE, *или* DROP.
4. Это выражение вычисляется относительно версии значения переменной $query_text, состоящего из символов верхнего регистра. Вы не только не изменяете значение $query_text, но даже не испытываете потребности в сохранении версии с символами в верхнем регистре. Если вам где-нибудь дальше опять понадобится версия с символами в верхнем регистре, вы можете снова вызвать функцию strtoupper.
5. Вы знаете, что preg_match при отсутствии соответствий возвращает 0, а PHP рассматривает 0 как false. Если есть соответствие, функция preg_match возвращает значение 1, которое PHP рассматривает как true. Поэтому вы можете

просто вставить весь вызов функции preg_match в условие вашей инструкции if и знать, что при наличии соответствия код инструкции if будет запущен и при его отсутствии этот код запущен не будет.

6. Внутри инструкции if переменной $return_rows присваивается значение false, поскольку соответствие означает, что запрос не возвращает строк.

Этот код не только проще читается и лучше воспринимается, но, кроме того, мы сократили 20 строк кода до 4.

ВНИМАНИЕ

Меньшее количество строк кода не всегда является плюсом. Порой ради экономии нескольких строк в жертву приносятся удобочитаемость и разборчивость кода, что не идет ему на пользу. Но когда есть возможность свести четыре или пять условий в одно или два, обычно это приносит положительный эффект.

Учет позиции

Одной из проблем даже в этой рационализированной версии сценария run_query.php является то, что она ищет соответствие во всем, что ввел пользователь. Если вы читали последнюю врезку в главе 4, то вам уже известно, что здесь кроются проблемы. Вам нужно избавить строку пользовательского запроса от лидирующих и замыкающих пробелов, что делается очень просто:

```
if (preg_match("/(CREATE|INSERT|UPDATE|DELETE|DROP)/",
               trim(strtoupper($query_text)))) {
  $return_rows = false;
}
```

Но есть еще одна непростая проблема: вам нужно искать эти специальные ключевые слова только в начале строки запроса. Это предохранит сценарий от сбоев на запросах, похожих на этот:

```
SELECT *
  FROM registrar_activities
 WHERE name = 'Update GPA'
    OR name = 'Drop a class'
```

Сбои возникают из-за ошибочного принятия за запросы с ключевыми словами UPDATE или DROP. Это запрос осуществляет выборку с помощью команды SELECT и возвращает строки, но если он интерпретируется как UPDATE или DROP, сценарий не покажет возвращенных строк.

Чтобы заставить сценарий работать должным образом, ранее для инструкции if нужны были дополнительные условия, но это было в черные дни до прихода регулярных выражений. Теперь можно просто сказать PHP: «Мне нужно использовать это выражение, но только в *начале* той строки, в которой ведется поиск».

В завершение этого колдовства добавьте в начало строки поиска знак вставки (^), который говорит «с начала»:

```
// Есть соответствия
echo "Количество соответствий: " . preg_match("/^(Mr|Dr). Smith/",
                                              "Dr. Smith") . "\n";
// Нет соответствий
echo "Количество соответствий: " . preg_match("/^(Mr|Dr). Smith/",
                                              "  Dr. Smith") . "\n";
```

Итак, в первом случае `/^(Mr|Dr). Smith/` соответствует Dr. Smith, поскольку строка начинается с `"Dr. Smith"` (Mr. Smith также будет соответствовать). Но вторая строка не имеет соответствий, потому что ^ признает неподходящими лидирующие пробелы.

Возвращаясь к средству запуска запросов, нужно сделать следующее:

```
if (preg_match("/^(CREATE|INSERT|UPDATE|DELETE|DROP)/",
               trim(strtoupper($query_text)))) {
  $return_rows = false;
}
```

Вся разница заключается в одном небольшом знаке вставки. То же самое, но только в конце строки можно сделать с помощью знака доллара `$`: он требует соответствия не в начале, а в конце строки, в которой ведется поиск:

```
// Нет соответствий
echo "Количество соответствий: " . preg_match("/^(Mr|Dr). Smith$/",
                        ,                     "Dr. Smith  ") . "\n";
// Есть соответствия
echo "Количество соответствий: " . preg_match("/^(Mr|Dr). Smith$/",
                                              "Dr. Smith") . "\n";
```

ВНИМАНИЕ

Знаки ^ и $ должны находиться внутри открывающего и закрывающего обратного слэша (/). Если вы поставили знак, например, как в /^(Mr|Dr). Smith/$, PHP пожалуется на последний знак $, объявив его неизвестным модификатором. Такую ошибку легко допустить, но ее бывает довольно трудно отследить, если вы не понимаете, что вы сделали.

Итак, в первом случае соответствий нет, потому что регулярное выражение, в котором используется знак `$`, не допускает замыкающих пробелов в «Dr. Smith ». А в результате второй проверки соответствие обнаруживается, потому что в строке нет лидирующих пробелов (что соответствует части регулярного выражения `^(Mr|Dr)`) и нет замыкающих пробелов (что соответствует части регулярного выражения `Smith$`).

Если знак ^ стоит в начале выражения, а знак `$` — в конце этого выражения, значит, по сути, вы требуете точного соответствия не только тому, что находится внутри строки поиска, но и самой строке.

Это похоже на то, как если бы вы сказали, что строка поиска должна быть идентична регулярному выражению. Разумеется, если бы вы работали с настоящей идентичностью в PHP (с помощью оператора == или ===), у вас не могло бы быть такой замечательной инструкции `or`, реализуемой с помощью знака `|`, или любого из других замечательных средств, предлагаемых регулярными выражениями.

Избавление от trim и strtoupper

Продолжим упрощение кода с помощью средств, предлагаемых регулярными выражениями. Только что вы занимались приведением всех букв значения переменной $query_text к верхнему регистру, а затем вели поиск CREATE, INSERT и им подобных ключевых слов применительно к версии запроса с символами в верхнем регистре.

Но регулярные выражения могут быть и нечувствительны к регистру символов, и тогда им будет все равно, в какой из версий слов (с прописными или строчными буквами) искать соответствия. Для этого нужно к концу выражения сразу за закрывающим обратным слэшем добавить символ i:

```
// Есть соответствия
echo "Количество соответствий: " . preg_match("/^(MR|DR). sMiTH$/i",
                         ,                    "Dr. Smith") . "\n";
```

Это выражение выдает соответствие, несмотря на то, что регистры символов выражения и строки, в которой ведется поиск, не соответствуют друг другу. Следовательно, вы можете изменить поиск в сценарии run_query.php, чтобы воспользоваться этой возможностью:

```
$return_rows = true;
if (preg_match("/^(CREATE|INSERT|UPDATE|DELETE|DROP)/i",
               trim($query_text))) {
  $return_rows = false;
}
```

Теперь больше нет функции strtoupper, а в конце выражения стоит новый символ i. С этими изменениями запрос, показанный на рис. 5.1, будет успешно распознан как DROP, в результате выполнения которого строки не возвращаются. Таким образом, даже не добавляя с помощью регулярных выражений функциональных возможностей, вы существенно усовершенствовали код. Вы ведете поиск нужных фрагментов в исходном значении $query_text, не изменяя его для работы с поиском. Именно так все и должно быть: поиск в неизмененной строке ввода везде, где это только возможно.

А что насчет удаления лишних пробелов? Оказывается, вам не нужно применять функцию trim к значению переменной $query_string. Вместо этого в регулярном выражении нужно просто проигнорировать лидирующие пробелы.

Подумаем еще немного: когда ведется поиск, разве что-нибудь игнорируется? Нет. Каким бы ни был результат, вы хотите сказать примерно следующее.

1. Нужно начать с соответствия любому количеству пробелов, включая и случай их полного отсутствия.
2. Затем после некоторого неопределенного количества пробелов нужно искать (CREATE|INSERT|UPDATE|DELETE|DROP).

Следовательно, игнорируя эти пробелы в нашей конкретной ситуации — определяя принадлежность запроса к CREATE, или к UPDATE, или к чему-то еще, — вы фактически используете другой тип соответствия.



Рис. 5.1. Запрос, введенный в форму

Теперь вы уже знаете, как задать соответствие пробелу: его нужно просто включить в регулярное выражение. Выражение /^ Mr. Smith/ требует открывающего пробела. Строка «Mr. Smith» ему не будет соответствовать, а вот строка « Mr. Smith» — будет.

Но для этого требуется пробел. А как показать, что вас устроит наличие более одного пробела? Для этого нужен символ +. Он говорит: «То, что стоит непосредственно передо мной, может появляться любое количество раз»:

```
// Есть соответствия
echo "Количество соответствий: " . preg_match("/^ (MR|DR). sMiTH$/i",
                           ,                   " Dr. Smith") . "\n";
// Нет соответствий
echo "Количество соответствий: " . preg_match("/^ (MR|DR). sMiTH$/i",
                                               "     Dr. Smith") . "\n";
```

```
// Есть соответствия
echo "Количество соответствий: " . preg_match("/^ +(MR|DR). sMiTH$/i",
                                    "     Dr. Smith") . "\n";
```

Первое и второе выражения ведут поиск конкретно одного пробела, и если первая запись соответствует, то вторая, с несколькими лидирующими пробелам, нет. Но третье выражение допускает наличие любого количества пробелов, поэтому здесь также найдено соответствие.

А теперь попробуйте следующий код:

```
// Нет соответствий
echo "Количество соответствий: " . preg_match("/^ +(MR|DR). sMiTH$/i",
                                    "Dr. Smith") . "\n";
```

Вот это да! Очевидно, «любое количество пробелов» для знака + в действительности означает «любое ненулевое количество пробелов». Если вас устраивает полное отсутствие символа или его любое количество, следует использовать знак *:

```
// Есть соответствия
echo "Количество соответствий: " . preg_match("/^ *(MR|DR). sMiTH$/i",
                                    "Dr. Smith") . "\n";
```

Итак, теперь вы можете поискать пробелы внутри значения `$query_text` в сценарии `run_query.php` и избавиться от воздействия на строку ввода, пусть даже и временного:

```
$return_rows = true;
if (preg_match("/^ *(CREATE|INSERT|UPDATE|DELETE|DROP)/i",
               $query_text)) {
  $return_rows = false;
}
```

Поиск набора символов

Посмотрите на рис. 5.2. Сможет ли ваша версия сценария `run_query.php` обработать то, что введено в это текстовое поле?

Здесь могут быть лидирующие пробелы, о чем невозможно сказать, просто изучив иллюстрацию или даже посмотрев в браузере. Но даже если здесь нет лидирующего пробела, есть кое-что другое: символ перевода строки. Ваши любимые премудрые пользователи сделали нечто такое, о чем вы даже подумать не могли: перед вводом кода SQL несколько раз нажали клавишу Enter.

И неожиданно для вас регулярное выражение перестало находить соответствие `DROP`-запросу, несмотря на обработку лидирующих пробелов и решение вопросов с символами в верхнем регистре. Причина в том, что при нажатии клавиши Enter клавиатура выдает специальные символы: обычно либо `\n`, либо в некоторых ситуациях `\r\n`, либо, чтобы все стало еще интереснее, временами выдает просто `\r`.

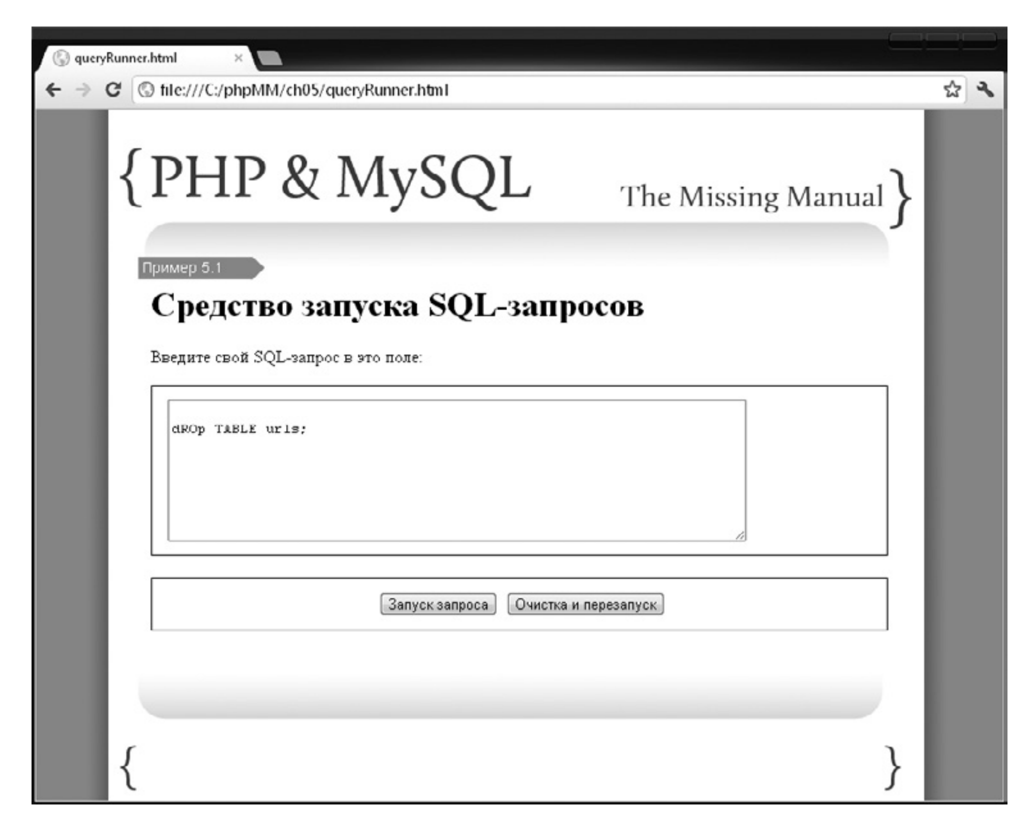


Рис. 5.2. Беспроблемный на первый взгляд запрос

ПРИМЕЧАНИЕ

Все они являются разновидностями символов новой строки: \n называется символом перевода строки, а \r — символом возврата каретки. В общем, в Windows используется \r\n, в Unix и Linux — \n, а в системах Mac (в частности, в старых операционных системах, предшествовавших OS X) — \r.

К счастью, с этими символами уже совсем не так много кросс-системных проблем, как это было всего несколько лет назад. Для создания новой строки вы совершенно свободно можете использовать \n, но при поиске иметь в виду все варианты.

Итак, что же вам делать? Учесть наличие нескольких таких символов довольно просто, например выражение `\n*` будет соответствовать любому количеству новых строк, а выражение `\r*` — любому количеству возвратов каретки.

А что делать с `\r\n`? Этому сочетанию будет соответствовать выражение `\r*\n*`. А как насчет пробелов? Можно воспользоваться выражением `\r*\n* *` и получить соответствие нажатию клавиши Enter с последующим набором пробелов, но если вы задумаетесь о пробелах и последующих нажатиях клавиши, а затем снова о пробелах... и новых нажатиях клавиши Enter, все снова усложнится.

Разумеется, самое главное предназначение регулярных выражений заключалось в устранении такого рода проблем. И вам это по силам: вы можете вести поиск

любого набора символов. Именно это вам и нужно: допустить любое количество (включая нулевое) любого набора символов, \r, \n или пробела. И при этом не заботиться ни о количестве появлений, ни о порядке следования.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Возвращение к отправной точке?

Может показаться, что вся эта работа с регулярными выражениями отбрасывает вас к той точке, с которой вы начали: к поиску CREATE, INSERT или UPDATE внутри $query_text. Если проигнорировать все лидирующие пробелы, не станет ли это тем же самым, что и выражение $location = strpos($query_text, "CREATE"); и все его связанные с if собратья?

Такое мнение может сложиться только на первый взгляд, но вы уже на тысячу километров отошли от всех этих инструкций if. Во-первых, вы уже получили сценарий, который не стыдно показать любому из ваших друзей-программистов. Вы воспользовались регулярными выражениями и распорядились ими с толком, поэтому у вас нет мусорного бака с условиями, в котором нужно копаться.

Во-вторых, ваш код стал более практичным. Он начинается с предположения, что будут возвращены строки. Затем на основании условия это предположение может измениться. Это вполне естественная логика: начать с одного пути, а если происходит что-нибудь еще, пойти по другому пути. Это намного лучше той разновидности обратной логики, которая была в ранней версии сценария run_query.php.

И самое важное: вы не ищете нужные ключевые слова SQL где-нибудь в $query_text. Поиск ведется в начале строки, с первого символа, не являющегося пробелом. Поэтому запрос такого вида:

```
SELECT *
  FROM registrar_activities
 WHERE name = 'Update GPA'
    OR name = 'Drop a class'
```

будет понят как SELECT и не будет, например, ошибочно принят за запрос DROP. И вы сделали это без объемного, запутанного, невразумительного и трудночитаемого кода.

Вы можете использовать что-нибудь вроде выражения (\r|\n|)*, в котором применяется вертикальная черта |, опять представляющая оператор or (или), а затем ко всей группе применяется символ *. Но когда вы имеете дело с одиночными символами, можно опустить знак | и просто поместить все допускаемые символы в набор, который обозначается квадратными скобками ([и]).

```
$return_rows = true;
if (preg_match("/^[ \t\r\n]*(CREATE|INSERT|UPDATE|DELETE|DROP)/i",
               $query_text)) {
  $return_rows = false;
}
```

Этот код справляется с пробелами, двумя разновидностями символов новой строки, и к нему добавлен символ \t для работы с символами табуляции. Итак, количество введенных лидирующих пробелов, символов табуляции или новых строк для вашего регулярного выражения значения не имеет, оно в состоянии справиться с ними. В действительности подобного рода пробельные символы носят настолько общий характер, что в регулярных выражениях можно использовать в качестве аббревиатуры выражения [\t\r\n] символ \s.

Благодаря этому код можно упростить еще больше:

```
$return_rows = true;
if (preg_match("/^\s*(CREATE|INSERT|UPDATE|DELETE|DROP)/i",
               $query_text)) {
  $return_rows = false;
}
```

Посмотрите, как этот код работает. Введите SQL-код, показанный ранее на рис. 5.2, и оправьте запрос. Регулярное выражение готово к обработке. Но, по всей видимости, вы почему-то получили нечто похожее на то, что показано на рис. 5.3. Что произошло?

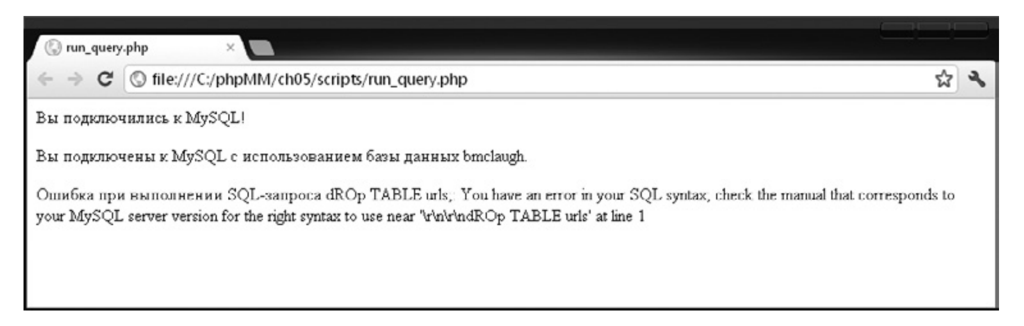


Рис. 5.3. Как только вы повысили надежность своего регулярного выражения и кода поиска, возникла новая ошибка, с которой придется разбираться

Возникшая проблема не имеет отношения к вашему регулярному выражению. На самом деле она заключается в том, что вы пытаетесь передать функции mysql_query некие запросы, которые не слишком хорошо были защищены от проблем подобно всем этим лишним последовательностям \r\n в начале запросов.

На самом деле существует множество запросов, которые будут создавать проблемы для функции run_query.php, независимо от качества кода вашего регулярного выражения. Попробуйте ввести следующий запрос:

```
SELECT *
  FROM urls
 WHERE description = 'home page'
```

Этот запрос кажется достаточно простым, но он также выводит из строя ваш сценарий. Наличие или отсутствие данных в таблице urls не играет роли, вы

по-прежнему будете получать ошибку (рис. 5.4). Пусть эта ошибка не вводит вас в заблуждение. Она не имеет никакого отношения к вашему коду SQL, просто у вашего сценария слишком примитивный код. Но не стоит волноваться, с таким хорошим подспорьем, как регулярные выражения, вы готовы справиться с более тесной интеграцией PHP и MySQL.

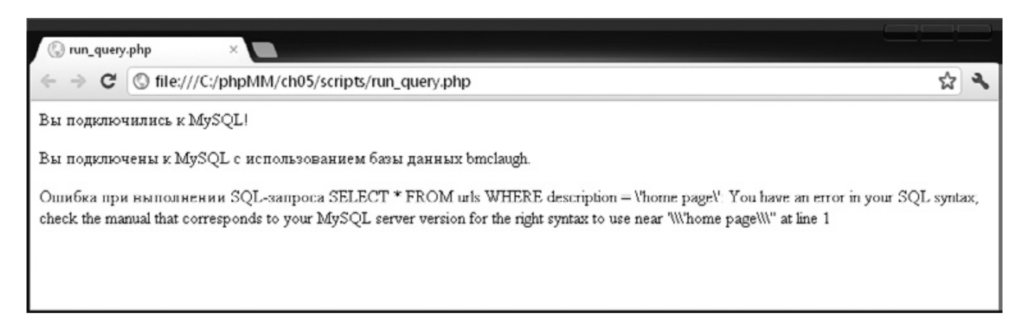


Рис. 5.4. Вывод ошибки

Если честно, на написание кода, требуемого для реальной обработки каждого возможного SQL-запроса и на обеспечение допустимости правильных и недопустимости неправильных элементов и на обработку всех типов запросов, у вас ушла бы не одна неделя.

Но эта затея не заслуживает внимания. Простой прием любых старых SQL-запросов — далеко не самая лучшая идея. Куда лучше отойти на шаг назад и подумать о том, что на самом деле нужно вашим пользователям. Наверное, это не пустая форма, и поэтому в следующей главе вы дадите им то, что им нужно: нормальную веб-форму, которая общается с MySQL на сервере базы данных.

Регулярные выражения: к бесконечности и еще дальше

Не покривлю душой, если скажу, что вы всего лишь слегка прикоснулись к регулярным выражениям. Хотя вы уже довольно твердо усвоили основы — от соответствия, ^ и $, различных вариантов `preg_match`, позиций и пробельных символов до +, * и наборов.

Но не нужно переживать или впадать в уныние и не стоит останавливаться в совершенствовании работы с PHP и MySQL и овладении регулярными выражениями. Мастерства добиться не так-то просто, и даже лучшие специалисты по регулярным выражениям углубляются в Google, чтобы вспомнить, как можно получить правильные последовательности символов, помещаемые между слэшами. Совершенство достигается практикой, поэтому не упускайте шансов воспользоваться регулярными выражениями. И по мере освоения PHP вы станете применять их все чаще. Постепенно вы ознакомитесь с ними не хуже самого PHP, HTML или любых других вещей, которыми плотно занимаетесь.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Регулярные выражения используются не только в PHP

Как вы уже, наверное, увидели, для освоения регулярных выражений нужно приложить некоторые усилия. В них используется немало странных символов, которые не только нужно найти на клавиатуре, но и научиться использовать. Поэтому регулярные выражения могут показаться вам сплошной тарабарщиной.

Но наградой за их освоение будет куда более широкая сфера применения, чем вы в состоянии себе представить. Например, в JavaScript также есть полная поддержка регулярных выражений. Регулярные выражения передаются в JavaScript таким методам, как `replace()`, а также методом `match()`, вызываемым в отношении строк. Поэтому все, что вы изучили в PHP, полностью можно перенести и в этот язык.

Вы также можете получить некоторые преимущества в HTML5. Регулярные выражения можно использовать в форме HTML5 для предоставления шаблонов, применяемых при проверке приемлемости данных. Следовательно, ваша работа в PHP поможет вам практически в любых аспектах веб-программирования.

Вряд ли есть какой-нибудь серьезный язык программирования, не поддерживающий регулярные выражения. Когда вы решите изучать Ruby и Ruby on Rails (а это должно произойти), вы уже будете уверенно разбираться в регулярных выражениях, они также очень пригодятся при переходе к использованию таких сред тестирования, как Cucumber, Capybara или TestUnit. Если все это звучит для вас несколько тревожно, не стоит волноваться! Вы уже всерьез занялись регулярными выражениями, даже еще не изучив, что собой представляет большинство этих языков.

Так какова же мораль всего этого повествования? То, что вы узнали о SQL, применимо не только в MySQL, а то, что вы узнали о регулярных выражениях, применяется не только в PHP. Ваше мастерство растет, поэтому используйте их где только можно.

6 Создание динамических веб-страниц

Мы уже создали довольно солидный набор инструментов. У нас есть PHP-сценарий для приема запросов из HTML-форм, MySQL для хранения информации от наших пользователей, регулярные выражения для манипулирования информацией в нужных нам форматах. А также такие элементарные средства управления ходом выполнения программы в PHP, как `if` и `for`, позволяющие создавать сценарии, которые принимают решения на основе предоставленной нашими пользователями информации.

Однако наша конечная цель изучения PHP и MySQL состоит в создании динамических и интересных веб-приложений. Пока мы еще мало что сделали в этом направлении. Вы получили несколько интересных форм, но даже они слишком просты: принимают информацию и выводят ее на экран, а также принимают SQL-запрос (причем справляются с этой задачей весьма посредственно).

К счастью, у нас есть все необходимое для того, чтобы приступить к созданию динамических страниц с использованием пользовательской информации и полноценных веб-приложений. Вы можете получить информацию от своих пользователей, сохранить ее в базе данных и даже провести элементарную обработку данных. Пришло время свести все это воедино в веб-страницы, которых ждут пользователи и которые позволят им вводить информацию, просматривать ее, а кроме того, дадут возможность просматривать сведения, связанные с пользователем.

Повторное обращение к пользовательской информации

В начале второй главы мы создали форму для ввода основного социального медийного профиля пользователя: идентификатора в Twitter, URL-адреса Facebook и некоторой основной контактной информации (см рис. 2.1). Итак, вы можете разрабатывать формы, взаимодействующие с PHP-сценариями и отправляющие

им данные тем же способом, которым вы создаете любую другую веб-страницу: вы используете HTML и CSS для создания аккуратной и понятной страницы. Затем привлекаете пользователей к посещению вашей страницы, заполнению полей и нажатию кнопок. И здесь на первый план выступает работа, выполняемая PHP и MySQL за сценой.

При этом обнаруживается требуемый объем работы: переход от простой формы в Интернете к сценарию, взаимодействующему с базой данных. Вам нужно определить, сконструировать и создать таблицы, наладить взаимодействие с этими таблицами, быть готовыми разобраться с ошибками, выдаваемыми базой данных, и это еще не все. Приступим к работе.

ПРИМЕЧАНИЕ

Если вы еще не сделали этого, скопируйте ранее упомянутую веб-форму HTML в свой рабочий каталог. Имя файла можете оставить прежним, но можете и переименовать его в create_user.html.

Действительно нет необходимости именять форму. А вот сценарий, получающий эту информацию, слишком примитивный. Все, что он делает, заключается в небольшой работе с текстом и его возврате на экран (рис. 6.1). Здесь и предстоит поработать, заставив сценарий *совершать определенные действия* над пользовательской информацией. HTML на рис. 6.1 сгенерирован вашим старым сценарием `getFormInfo.php`. Он имеет неприглядный вид и определенно нуждается в улучшении. Беспокоит еще и то, что эта информация нигде не сохраняется. Как только пользователь уходит с сайта, его информация теряется.

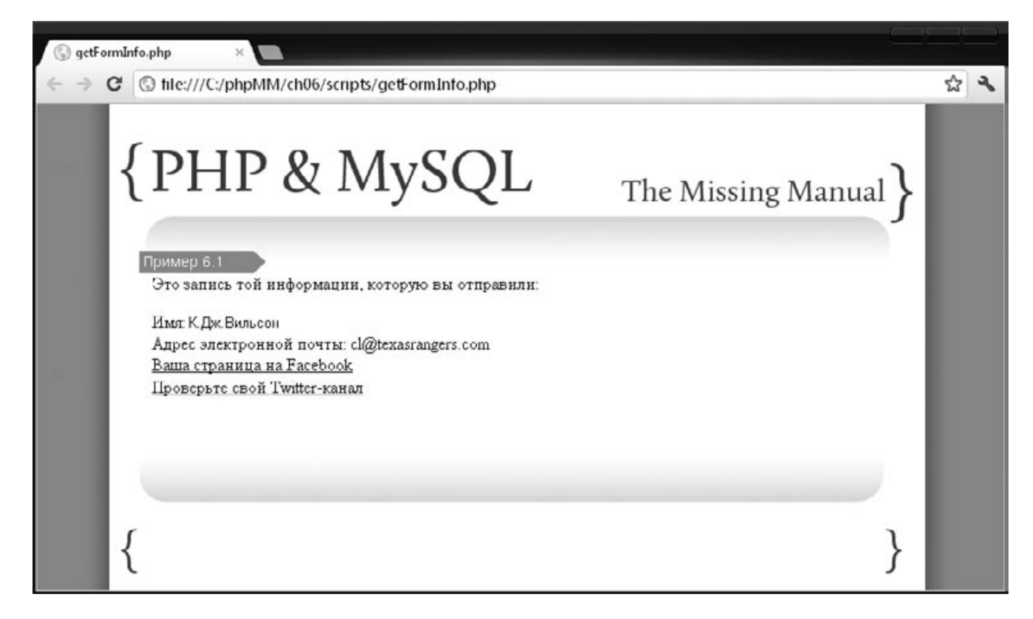


Рис. 6.1. Результат выполнения сценария getFormInfo.php

Проектирование таблиц базы данных

Создание веб-приложений во многом похоже на прохождение сложного лабиринта: иногда сложнее всего решить, с чего начать. Обычно веб-форме нужен сценарий, которому она может отправлять данные. Однако этому сценарию необходима таблица, в которую он сможет вставлять информацию. Но где эта таблица? В базе данных, которую вам нужно создать или настроить на доступ через Интернет. А затем следует заняться самой таблицей: ей нужна структура. И этот путь проходит почти каждая форма любого приложения: то, что начинается со страницы, которую обычно видит пользователь, зачастую заканчивается структурой сервера базы данных, не видимой никому, кроме программиста.

Всегда проще начать с той информации, которую нужно сохранить. Кое-что уже сделано при создании формы ввода (см. рис. 2.1). Итак, на данный момент вы собираете следующую информацию, предоставляемую вашими пользователями:

- имя;
- фамилия;
- адрес электронной почты;
- URL-адрес Facebook;
- идентификатор в Twitter.

Все действительно очень просто: вам нужны лишь эти пять фрагментов информации, каждый из которых касается пользователя. Поэтому нетрудно прийти к заключению, что вам нужна таблица для хранения сведений о пользователях и у каждого пользователя есть имя, фамилия, адрес электронной почты, URL-адрес Facebook и идентификатор в Twitter.

Теперь просто переведите это в SQL-инструкцию CREATE, с чем у вас не должно быть проблем:

```
CREATE TABLE users (
       user_id int,
       first_name varchar(20),
       last_name varchar(30),
       email varchar(50),
       facebook_url varchar(100),
       twitter_handle varchar(20)
);
```

ВНИМАНИЕ

Может быть, не нужно пока погружаться в окно командной строки MySQL или веб-формы и запускать эту команду. Лучшие времена для них пока не наступили, поскольку вскоре ожидаются весьма важные дополнения.

Возможно, вы помните этот SQL-код по главе 3. Это было целую вечность назад, когда у вас еще были весьма приблизительные представления о базах данных. Теперь вы точно знаете, что должно попадать в эту таблицу: информация из веб-формы, которая у вас уже есть.

К ВАШЕМУ СВЕДЕНИЮ

Здесь одно похоже на другое

Если пообщаться со специалистом по базам данных, вы сразу же столкнетесь с массой взаимозаменяемых понятий.

В таблицах есть строки, и каждый элемент в таблице является *строкой*. Но вы также услышите, что строка, называемая *элементом таблицы*, еще называется *записью*. Все эти термины означают одно и то же, и при этом технически грамотнее, наверное, сказать, что у таблицы есть строки, а не элементы или записи, но в реальной жизни эти термины используются как равнозначные.

В том же духе поля в таблице, например `first_name` или `last_name`, называются также *столбцами*. И в этих полях (или столбцах) у вас имеются *значения*, или *информация*, или, в скучной технической терминологии, *данные*. Масса разных терминов, означающих одно и то же.

Лучше, конечно, избегать смешанных сочетаний этих терминов. Строки и столбцы — в одну сторону, а записи и поля — в другую. У таблицы со строками есть столбцы; а у таблицы с записями есть поля.

Прежде всего следует помнить, что, как и в случае со всеми другими элементами языка, бал правит контекст. Намного важнее знать, что вписывается в контекст, а что нет, чем следить за тем, что вы произнесли слово «строка» вместо слова «запись». И не путайте весь элемент с его отдельными частями. Например, отдельный элемент, запись или строка в таблице `users` обладает несколькими полями, столбцами или частями информации. Усвойте все это, и вы сможете разобраться со всем остальным, внимательно слушая собеседника и задавая правильные вопросы.

В правильных таблицах баз данных всегда есть столбец id

Следует также обратить на внимание на имеющееся в таблице поле `user_id`. Подумайте о наиболее частых действиях с базой данных. Это не создание новых записей или профилей пользователя, а поиск нужной информации и *доступ* к ней.

А теперь подумайте, как вы ищете информацию? Вы можете вести поиск по фамилии и затем находить соответствующие записи или вести поиск по адресу электронной почты или по идентификатору в Twitter, который предполагается уникальным для каждого пользователя. Фактически вас, наверное, часто просили создать уникальное имя пользователя (и зачастую для этого приходилось постараться, поскольку в Twitter осталось не так уж и много «нормальных» имен, если не считать таковым m97f-ss0).

Базы данных не составляют исключения: им нужно что-нибудь, что можно будет искать. Более того, базы данных хорошо работают только в том случае, когда они могут идентифицировать каждую отдельную строку в таблице с помощью уникального фрагмента информации. Но для базы данных нужно учесть еще одно обстоятельство: базы данных лучше работают с числами, чем с текстом. Для строки в таблице абсолютно предпочитаемым типом уникального идентификатора, или ID, является уникальное число.

Именно для него и предназначено поле user_id. Оно содержит специальное уникальное значение для каждой строки, и это значение числовое. Оно идентифицирует каждого пользователя как непохожего ни на кого другого и позволяет базе данных работать быстро и успешно.

Ваш друг автоприращение

Но здесь, в зарослях SQL, кроется небольшая проблема. Если назначение поля user_id заключается в предоставлении уникального идентификатора для каждого пользователя, кто будет выполнять задачу поддержания уникальности этого ID? Как все сценарии (а их к завершению вашего большого веб-приложения будет не один и не два) смогут гарантировать, что никакие два пользователя не вводят данные с одним и тем же user_id в таблицу users?

Это не такая простая проблема, поскольку, если вы утратите возможность уникальной идентификации пользователя, дела быстро покатятся вниз. С другой стороны, никому не хочется тратить много времени на написание генератора чисел для каждой таблицы или для каждого создаваемого веб-приложения.

Решение кроется не в коде, а в базе данных. Большинство баз данных, и MySQL в том числе, дают вам возможность использования автоприращения. Вы указываете это значение для поля, и при каждом добавлении строки к таблице в поле автоматически создается новое число, имеющее приращение по сравнению с числом в предыдущей добавленной строке. Поэтому, если один сценарий добавил нового пользователя и MySQL установила для user_id значение 1029, а затем другой сценарий добавил нового пользователя, MySQL просто добавляет единицу, получает число 1030 и устанавливает его в качестве ID нового пользователя.

Это свойство можно добавить к инструкции создания таблицы CREATE:

```
CREATE TABLE users (
       user_id int AUTO_INCREMENT,
       first_name varchar(20),
       last_name varchar(30),
       email varchar(50),
       facebook_url varchar(100),
       twitter_handle varchar(20)
);
```

Уже лучше. Теперь не нужно волноваться за числа ID. Фактически вам не нужно делать что-нибудь особенное, чтобы заставить MySQL заполнить столбец user_id. При каждом добавлении новой строки MySQL также добавляет новое значение для user_id.

ID и первичные ключи — хорошие компаньоны

После установки автоприращения значения user_id в MySQL обычно делается еще одна тонкая вещь: в качестве первичного ключа таблицы users определяется user_id. *Первичный ключ* — это понятие базы данных для специального уникального значения, имеющегося у каждой таблицы.

ПРИМЕЧАНИЕ

В некоторых особых случаях можно создать первичный ключ на основе сразу нескольких столбцов, но обычно этого не делают.

Первичные ключи играют важную роль, поскольку базы данных обычно создают индекс на основе первичного ключа таблицы. *Индекс* — это механизм базы данных, с помощью которого она может быстро найти строки на основе этого индекса. Если у вас есть проиндексированный столбец user_id, вы можете найти строку со значением user_id, равным 2048, намного быстрее, чем при поиске строки с таким же user_id, но в таблице, где столбец user_id не проиндексирован.

Проиндексированное поле похоже на высокоорганизованный набор значений, позволяющий проводить быстрый поиск. Поиск может также вестись и в непроиндексированном поле, но тогда база данных должна последовательно перебирать каждое значение, пока не будет найдено точно то значение, которое вы ищете. Представьте себе, например, поиск книги в хорошей библиотеке и на захламленном чердаке дома вашего прадедушки.

Заставляя MySQL применить к полю user_id автоприращение — AUTO_INCREMENT user_id, — вы идентифицируете это поле как имеющее особый статус. Фактически MySQL не даст вам установить AUTO_INCREMENT более чем для одного поля, поскольку она предполагает, что это свойство задано для поля, используемого в качестве первичного ключа.

Но здесь нужно кое-что запомнить. Можно подумать, что это «кое-что» MySQL сделает за вас, но она не будет этого делать. Вы должны сказать MySQL, что хотите сделать user_id первичным ключом:

```
CREATE TABLE users (
       user_id int AUTO_INCREMENT PRIMARY KEY,
       first_name varchar(20),
       last_name varchar(30),
       email varchar(50),
       facebook_url varchar(100),
       twitter_handle varchar(20)
);
```

Эта строка делает явным то, что было неявным при установке свойства AUTO_INCREMENT: поле user_id становится уникальным идентификатором для каждой записи пользователя в вашей таблице. Если вы не добавите эту строку, MySQL выдаст ошибку. Предположим, что данный код SQL не содержит строки PRIMARY KEY:

```
CREATE TABLE users (
       user_id int AUTO_INCREMENT,
       first_name varchar(20),
       last_name varchar(30),
       email varchar(50),
       facebook_url varchar(100),
       twitter_handle varchar(20)
);
```

Если вы запустите этот запрос, MySQL выдаст странную ошибку, которая показана на рис. 6.2 в консоли phpMyAdmin.



Рис. 6.2. Ошибка в phpMyAdmin

ПРИМЕЧАНИЕ

phpMyAdmin является отличным средством для запуска запросов. Вам не придется тратить уйму времени на средство, работающее в текстовом режиме, и вы сможете просматривать свои таблицы визуально. Большинство компаний, занимающихся веб-хостингом и предоставляющих MySQL и службы баз данных, предлагают phpMyAdmin. Это означает, что, ознакомившись с этим средством на одном хосте, вы, вероятно, сможете работать с таким же средством и на другом хосте.

Ошибка #1075, печально известная тем, кому приходилось долгое время работать с MySQL, сообщает вам, что, поскольку вы добавили столбец с функцией AUTO_INCREMENT, вы должны пометить его как PRIMARY KEY. Было бы неплохо, если бы MySQL сама позаботилась об этом за вас, но, увы, пока это ваша задача. Поэтому верните назад PRIMARY KEY, и вы будете практически готовы без ошибок создать эту таблицу для реального использования.

Добавление ограничений к базе данных

Предназначение таких полей, как user_id, заключается в облегчении поиска. Добавление AUTO_INCREMENT (и установка поля в качестве первичного ключа) помогает поиску, но при создании столбца со свойством AUTO_INCREMENT происходят и другие совершенно неочевидные вещи. Добавляя это свойство, вы словно говорите: «У этого поля всегда будет вполне определенное значение». Дело в том, что MySQL заполняет это поле сама.

Но есть и другие поля, из которых вы хотите получить информацию, например поля имени и фамилии. И возможно, вы потребуете также заполнить адрес электронной почты. Идентификаторы в Twitter и Facebook бывают у пользователей не всегда, поэтому соответствующие поля можно не заполнять, но все остальное обязательно для заполнения.

Вы можете даже позволить PHP-сценариям и веб-страницам потребовать от пользователя эту информацию. Но надежно ли это? Что если кто-нибудь еще забудет добавить проверку на веб-страницу? Что если вы сами забудете об этом, например, когда будете создавать код сильно уставшим? Проверку приемлемости данных нужно внедрять везде, где это только возможно.

И тут опять MySQL предоставляет вам все, что нужно. Вы можете потребовать наличия значения в поле, сообщив MySQL, что поле не может оставаться пустым со значением NULL, означающим в переводе с программистского языка «нет значения»:

```
CREATE TABLE users (
       user_id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
       first_name varchar(20) NOT NULL,
       last_name varchar(30) NOT NULL,
       email varchar(50) NOT NULL,
       facebook_url varchar(100),
       twitter_handle varchar(20)
);
```

ПРИМЕЧАНИЕ

Даже притом, что обработчики MySQL сами осуществляют автоприращение и вставку значения в поле user_id, все же будет неплохо придать этому полю свойство NOT NULL. Это вполне определенно говорит о необходимости наличия значения в этом поле, независимо от того, как MySQL или любой другой код на самом деле заполняют это поле.

Как и в случает со свойством AUTO_INCREMENT, это изменение вносится быстро, просто и играет важную роль в защите целостности вашей информации (или, если говорить точнее, информации вашего пользователя!).

ЗАМЕТКИ О ПРОЕКТИРОВАНИИ

NULL или NOT NULL

Хотя в таблице users определить, какие столбцы должны иметь свойство NOT NULL, не составило труда, бывают и другие случаи. На самом деле даже с таблицей users не все так однозначно: вы уверены, что нужно затребовать адрес электронной почты? Вполне возможно, что у кого-то его не окажется (хотя зачем пользователи, у которых нет такого адреса, бродят по Интернету, для меня большая загадка). А некоторые опасаются спама и не желают вводить адрес электронной почты. Вы уверены, что хотите затребовать его в качестве части пользовательской информации?

Придание столбцу свойства NOT NULL — одно из самых важных решений, принимаемых вами в отношении отдельной таблицы. В частности, это касается решения *не* придавать столбцу свойства NOT NULL. Любая добавляемая запись может иметь в таком столбце значение NULL, и если вы в конечном счете решите: «Ой, а ведь мне действительно нужно, чтобы здесь было значение», помехой станут все старые записи, у которых этого значения нет.

> Но не стоит слишком увлекаться присваиванием столбцам свойства NOT NULL. Возможно, вы подумаете, что будет надежнее почаще применять это свойство и быть уверенным, что получите больше, а не меньше данных. Однако пользователи могут расстроиться от перспективы обязательного заполнения 28 полей только для того, чтобы пользоваться вашим сайтом. Даже такие мегасайты, как Facebook и Twitter, требуют ввода минимального объема информации: обычно это имя, адрес электронной почты, имя пользователя и пароль.
>
> В общем, за правило нужно взять следующее: необходимо требовать только те данные, без которых никак не обойтись, но требовать их неукоснительно. Обдумайте все хорошенько и примите правильное решение.
>
> Конечно, недовольные все равно найдутся. Ваша цель — угодить большинству своих пользователей в большинстве случаев. Если вы можете осуществить это все, еще получая от них нужную вам информацию, значит, вы на верном пути.
>
> И еще одна небольшая подсказка: со свойством NOT NULL вы работаете на уровне таблицы, а не на уровне приложения. Иными словами, вы говорите следующее: «Этот столбец не может иметь значение null, если (и только если) это касается записи в данной таблице». Поэтому вы можете решить, что пользователям необязательно вводить адрес, но если они ввели адрес, требуется набрать улицу, город и страну. Если будете думать о том, какие данные имеют значение конкретно для этой таблицы, а не для всего приложения, это поможет вам задать ограничения в базе данных, позволяющие вводить по-настоящему ценные данные и не проявлять ненужного усердия в присвоении столбцам свойства NOT NULL.

У вас должна получиться исключительно полезная инструкция SQL, поэтому нужно продолжить работу и создать таблицу users. Войдите в MySQL с помощью окна командной строки созданной ранее веб-формы или с помощью другого веб-средства, наподобие phpMyAdmin, и создайте таблицу. Вскоре она вам пригодится.

ВНИМАНИЕ

Вам может понадобиться с помощью команды DROP удалить прежнюю версию таблицы. Если при попытке создания таблицы выдается ошибка, можно просто воспользоваться командой DROP TABLE users;. Она очистит базу данных от любой существующей версии таблицы. Перед запуском инструкции CREATE убедитесь в том, что вы находитесь в нужной базе данных.

Сохранение информации о пользователе

У нас уже была таблица, а теперь мы получили версию таблицы users, усиленную свойством AUTO_INCREMENT и проверкой наличия значений в некоторых ключевых полях. Кроме того, наша веб-форма собирает именно ту информацию, которая нужна для наполнения таблицы. Осталось только связать все это вместе с помощью PHP. У нас уже есть практически все, что для этого нужно.

Вы можете приступить к созданию нового сценария или воспользоваться в качестве отправной точки своей старой версией getFormInfo.php. В любом случае сначала нужно забрать введенную пользователем информацию и немного обработать текст, чтобы получить желаемые значения:

```
<?php

$first_name = trim($_REQUEST['first_name']);
$last_name = trim($_REQUEST['last_name']);
$email = trim($_REQUEST['email']);
$facebook_url = str_replace("facebook.org", "facebook.com", trim($_
REQUEST['facebook_url']));
$position = strpos($facebook_url, "facebook.com");
if ($position === false) {
  $facebook_url = "http://www.facebook.com/" . $facebook_url;
}

$twitter_handle = trim($_REQUEST['twitter_handle']);
$twitter_url = "http://www.twitter.com/";
$position = strpos($twitter_handle, "@");
if ($position === false) {
  $twitter_url = $twitter_url . $twitter_handle;
} else {
  $twitter_url = $twitter_url . substr($twitter_handle, $position + 1);
}

?>
```

Назовите этот сценарий create_user.php и сохраните его файл в каталоге scripts/, который нужно поместить либо в корневом каталоге вашего сайта, либо в подкаталоге ch06/ каталога examples. Необходимо будет также обновить действие в форме сценария create_user.html, чтобы данные отправлялись сценарию с этим только что обновленным именем.

Вы уже вводили этот код, и поскольку изменения в форму не вносились, он сохранил работоспособность. Теперь следует лишь обновить его, чтобы он сохранял эту информацию в вашей новой таблице users.

ПРИМЕЧАНИЕ

В качестве самостоятельного задания попробуйте переделать сценарий create_user.php, чтобы для обновления используемых там переменных вместо функции strpos использовались регулярные выражения.

Создание SQL-запроса

Сначала вы можете воспользоваться своим уже существующим сценарием подключения к базе данных, чтобы не испытывать трудностей с подключением:

```
<?php

require '../../scripts/database_connection.php';

// помещение запрошенной пользовательской информации в переменные

?>
```

ВНИМАНИЕ

У вас в database_connection.php могли остаться инструкции echo (из ранних версий примеров). В таком случае удалите их сейчас, чтобы они не нарушали целостности восприятия от информации, предоставляемой вашим пользователям.

Когда подключение к базе данных будет готово к использованию, вы должны превратить всю имеющуюся информацию в желаемую инструкцию вставки INSERT, чтобы можно было поместить информацию в вашу базу данных. Но не будем углубляться в код и начнем с простой инструкции. Например, возьмите набор случайных значений (возможно, свои собственные данные) и создайте SQL для выполнения поставленной задачи:

```
INSERT INTO users (first_name,
                   last_name,
                   email,
                   facebook_url,
                   twitter_handle)
           VALUES ("Brett",
                   "McLaughlin",
                   "brett.m@me.com",
                   "http://www.facebook.com/bdmclaughlin",
                   "@bdmclaughlin");
```

ПРИМЕЧАНИЕ

Для проверки этого SQL-кода на работоспособность и приемлемость использованного формата можно воспользоваться имеющимся у вас инструментарием MySQL.

Эта инструкция станет своеобразным шаблоном: вам нужна ее основа. Необходимо будет заменить значения, использующиеся в ней в качестве образца, информацией, которая будет запрошена у пользователя. Поскольку вы уже получили эти значения, данная задача не потребует больших усилий:

```
$insert_sql = "INSERT INTO users (first_name, last_name, " .
                          "email, facebook_url, twitter_handle) " .
              "VALUES ('{$first_name}', '{$last_name}', '{$email}', " .
                      "'{$facebook_url}', '{$twitter_handle}');";
```

ПРИМЕЧАНИЕ

Вам, возможно, не понадобится разбивать данный код на такое большое количество отдельных строк, объединенных точками. Это было необходимо для того, чтобы он поместился по ширине книжной страницы.

В этом примере мы создаем новую строковую переменную, содержащую SQL-запрос. Затем вы можете передать этот запрос функции mysql_query и запустить ее применительно к вашей базе данных. Обратите внимание, что каждое посланное вами в базу данных значение, которое должно попасть в текстовое поле таблицы users, должно быть взято в кавычки. Применение одинарных кавычек позволяет воспользоваться двойными кавычками, чтобы заключить в них весь запрос, а задействование фигурных скобок { } дает возможность вставить значения переменных непосредственно в строку запроса.

Вставка данных о пользователе

Теперь настало время добавить пользователя к нашей базе данных, и это, пожалуй, самая простая в написании (и зачастую самая забавная) строка PHP-кода, запускающего код SQL:

```
<?php

// Обработка запроса пользователя

$insert_sql = "INSERT INTO users (first_name, last_name, email,
facebook_url, twitter_handle) " .
              "VALUES ('{$first_name}', '{$last_name}', '{$email}', " .
                      "'{$facebook_url}', '{$twitter_handle}');";

// Вставка данных о пользователе в базу данных
mysql_query($insert_sql);

?>
```

Отлично! Но это еще не все. Нужно принять в расчет возможные ошибки. Например, что если вы сначала забудете добавить таблицу users? Или у вас уже есть таблица users, но без столбца facebook_url, с неправильно названным или с неправильно написанным именем столбца?

Кода дело доходит до отчета об ошибках, здесь есть над чем потрудиться. Но сейчас мы воспользуемся очень простым (наверное, даже слишком простым) подходом:

```
<?php

// Обработка запроса пользователя

$insert_sql = "INSERT INTO users (first_name, last_name, email,
facebook_url, twitter_handle) " .
              "VALUES ('{$first_name}', '{$last_name}', '{$email}', " .
                      "'{$facebook_url}', '{$twitter_handle}');";

// Вставка данных о пользователе в базу данных
mysql_query($insert_sql)
  or die(mysql_error());

?>
```

Использование метода die далеко от идеала, но по крайней мере это работает и выдает хоть какой-то отчет об ошибке.

А теперь вы можете испытать в деле свою страницу, правда, не особо при этом стараясь. Итак, зайдите на свою веб-страницу и заполните ее данными образца, показанного на рис. 6.3.

Теперь отправьте страницу, и ваш новый код будет запущен. Он сконструирует инструкцию SQL, используя ваши значения, подключится к базе данных и вставит данные с помощью функции mysql_query. К счастью, инструкция die запущена не будет.

Рис. 6.3. Заполняем страницу данными

Если предположить, что никаких ошибок не возникло, форма не вернет практически ничего. Обескураживающий результат, но что-то ведь произошло, особенно если учесть, что вы не получили сообщения об ошибке.

ПРИМЕЧАНИЕ

Если у вас по-прежнему остался HTML-раздел из сценария getFormInfo.php, скопированный в сценарий create_user.php, то после отправки формы в ответ могла быть выведена какая-нибудь информация.

Чтобы определить, что произошло в вашей базе данных, запустите инструментарий SQL и введите следующий запрос:

```
SELECT user_id, first_name, last_name
  FROM users;
```

В ответ вы должны получить нечто такое:

```
+---------+------------+-----------+
| user_id | first_name | last_name |
+---------+------------+-----------+
|       1 | C. J.      | Wilson    |
+---------+------------+-----------+
1 row in set (0.00 sec)
```

Если используется phpMyAdmin, вы можете перейти к своей таблице `users` и проверить любые данные, которые могут находиться внутри этой таблицы (рис. 6.4).

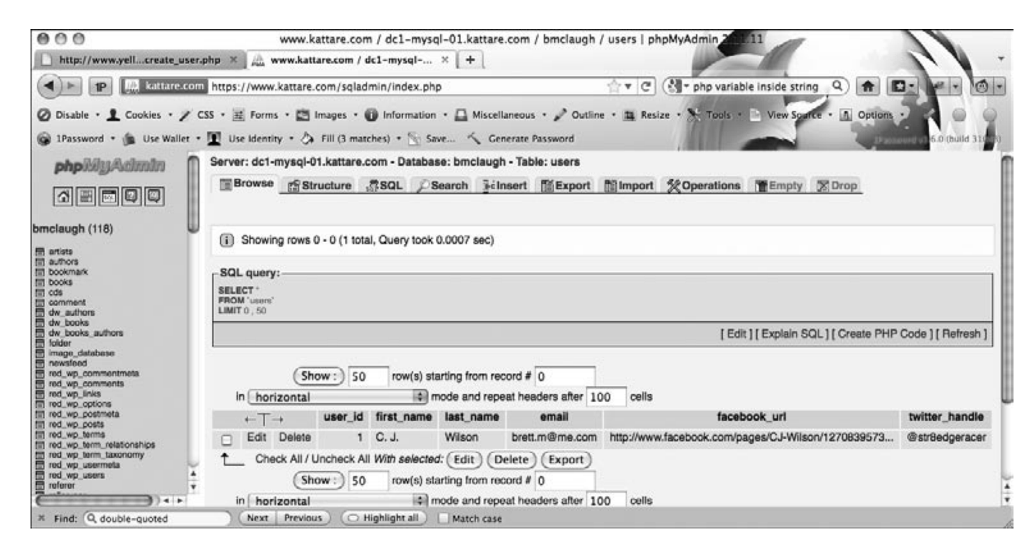


Рис. 6.4. Пример данных внутри таблицы users

Здесь вы можете увидеть, что запись в таблице `users` имеет не только данные, извлеченные из веб-формы, но также сгенерированное в автоматическом режиме (и получившее автоприращение) значение идентификатора: в данном случае оно равно 1. По мере увеличения количества пользователей это число будет продолжать расти, хотя нельзя рассчитывать на то, что оно будет получать последовательные значения.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Имена должны соответствовать выполняемым действиям

Когда у вас то там, то здесь есть лишь несколько веб-страниц, задумываться над именами особо не приходится. Каким будет имя страницы, `getFormInfo.html` или `create_user.html`, в сущности, безразлично. Все ваши файлы могут поместиться на одной странице листинга каталога или в окне FTP-клиента.

Но даже в средних по объему веб-приложениях у вас будет значительно больше файлов. Если вы приступите к обязательному тестированию, у вас запросто может оказаться несколько сотен файлов. В такой ситуации вы станете испытывать реальную потребность в именах, имеющих вполне определенный смысл.

Но значение, заложенное в имя, должно быть не только простым описанием. Многие ваши формы и сценарии будут относиться непосредственно к отдельной таблице в базе данных и фактически совершать одно конкретное действие по отношению к этой таблице, например создание записи о пользователе в таблице `users`.

> В таких случаях вы действительно сможете облегчить жизнь себе и другим, которые будут работать с вашим кодом, называя свои файлы в соответствии с их предназначением. Следовательно, если ваша форма может получать социальную сетевую информацию пользователя, она в конечном счете создает запись пользователя, и имя `create_user.php` является для нее достаточно описательным, простым и понятным.
>
> Кроме того, вы вскоре узнаете о CRUD — идее, предназначенной для информации любого типа, подобно информации о пользователе. Вы создаете (CReate), обновляете (Update) и удаляете (Delete) эту информацию. Проецирование ваших HTML-страниц и сценариев на такие базовые действия (`create_user`, `update_user` и т. д.) действительно помогает вам увидеть, что у вас имеется и чего не хватает.

Первое действие по подтверждению

На данный момент вы получили запись о пользователе (или столько записей, сколько вы сделали с помощью вашей веб-формы), но пользователь вашей программы не видит ничего, кроме пустого экрана. Настало время задать какую-нибудь ответную реакцию, чтобы пользователи знали, что происходит после того, как они заполнили форму.

Для начала вы можете вернуться назад к коду из представленного ранее сценария `getFormInfo.php`:

```
<?php

// Получение информации о пользователе из массива request

// Подключение к базе данных и вставка пользователя

?>

<html>
 <head>
  <link href="../../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>
 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пример 6.1</div>
  <div id="content">
    <p>Это запись той информации, которую вы отправили:</p>
    <p>
      Имя: <?php echo $first_name . " " . $last_name; ?><br />
      Адрес электронной почты: <?php echo $email; ?><br />
      <a href="<?php echo $facebook_url; ?>">Ваша страница на Facebook</a>
      <br />
      <a href="<?php echo $twitter_url; ?>">Проверьте свой Twitter-канал</a>
```

```
        <br />
      </p>
    </div>

    <div id="footer"></div>
  </body>
</html>
```

Эта страница лучше, чем ничего, но вы можете увидеть шероховатости, которые можно исправить прямо сейчас. Вы не вывели идентификатор в Twitter, ограничившись выводом URL для этого идентификатора. Хотя это удобнее для щелчка, информация не отображает то, что было введено в базу данных. Таким образом, перед вами стоит нелегкий выбор.

- Вы можете вывести то, что было введено в базу данных, то есть значение переменной `$twitter_handle`. Это то, что было вставлено в текст, но оно не имеет особой ценности для веб-страницы, а просто позволяет вашим пользователем узнать, что находится в базе данных. А нужна ли им структура вашей базы данных?
- Вы можете вывести URL, более подходящий для щелчка, но не связанный напрямую с тем, что находится в базе данных. Такая модификация значения базы данных вполне приемлема, но она может не соответствовать в точности той форме, которая ориентирована на добавление записи о пользователе в базу данных.

Сейчас применительно к идентификатору в Twitter этот вопрос не имеет особой важности. Но аналогичный вопрос возникает по поводу вывода имени и фамилии или их объединения, как в этом коде:

```
Name: <?php echo $first_name . " " . $last_name; ?><br />
```

Здесь можно задаться и более серьезным вопросом: что именно вы показываете своим пользователям? Конкретные значения в том виде, в котором они хранятся в базе данных, или же вы обрабатываете данные, приводя их в более удобный для пользовательского восприятия вид?

Не нужно путать пользователей с программистами

Как предлагается в предыдущем примере, своим пользователям *всегда* нужно показывать то, что имеет для них определенный смысл. Пользователю вряд ли захочется узнать о столбцах в вашей базе данных, о значении первичного ключа или о том, в каком виде вы храните их идентификатор Twitter — со знаком `@` или без него. Сконцентрируйтесь на том, *что* ваши пользователи хотят видеть, а не на конкретном содержимом базы данных.

Но это еще не все. Что будет источником показываемой вами информации? Применительно к этому идея показа пользователю того, что имеет для него

определенный смысл, превращается для вас в идею изъятия информации из базы данных, ее обработки для приведения в надлежащий формат с последующим показом этой преобразованной информации пользователю.

Но разве при этом первом проходе при подтверждении наличия данных вы показываете то, что хранится в базе данных? Конечно нет, вы просто возвращаете пользователю то, что он вам предоставил. А если что-нибудь случилось при вставке информации в вашу таблицу базы данных? Вы об этом никогда не узнаете. Показывая пользователю его собственную информацию, вы можете скрывать то, что на самом деле попало в базу данных.

Так что же делать? Пользователям нужно показать что-то, имеющее для них определенный смысл. Но кроме того, необходимо отобразить все эти значения на основе данных, хранящихся в базе, а не на основе механического повторения формы, поскольку при этом не будут выявляться проблемы с базой данных.

Надеюсь, что вы справитесь с обеими задачами! Предположим, что есть способ извлечения пользовательской информации из базы данных (возможно, с использованием SQL-инструкции SELECT). А затем на основе этой информации, полученной из базы данных (которая отражает проблемы, если они есть), создать нечто, что пользователь сможет увидеть и прочитать и что будет иметь для него вполне определенный смысл.

Одним из решений может стать перезагрузка информации после ее ввода пользователем, например с помощью следующего кода:

```
<?php

// Получение информации о пользователе из массива request

// Подключение к базе данных и вставка пользователя

$get_user_query = "SELECT * FROM USERS WHERE ..."
mysql_query($get_user_query);

// Загрузка этой информации и ее подготовка к выводу в виде HTML

?>

<!-- HTML-вывод -->
```

ВНИМАНИЕ

Значение переменной $get_user_query в этом коде специально оставлено в незавершенном виде. Показанное троеточие не будет работать, вам нужно будет включить вместо него часть инструкции WHERE, указывающей на только что добавленного в базу пользователя.

Код дает вам пользователя из базы данных, позволяя при этом модифицировать полученные данные для их лучшего восприятия пользователем. Вам остается определить, как найти конкретного пользователя, который только что был добавлен, но это мы рассмотрим чуть позже.

Используя данный код, вы работаете с текстом информации запроса, а затем вам приходится опять подвергать его обработке на основе ответа от базы данных. Неужели это лучший способ решения задачи?

Конечно же, нет. Нужно подумать о приложении в целом: нет ли какого-нибудь другого места, где нужно показывать пользователя? Конечно, есть. В каждом приличном приложении имеется место, где пользователь может проверить свой собственный профиль. Для предоставления такой функциональной возможности нужно взять код из последней части сценария `create_user.php` и скопировать его в сценарий `show_user.php`. Но это не самая лучшая идея. Вспомним, что вам очень не хотелось иметь один и тот же код более чем в одном месте. Именно поэтому у вас появился впечатляющий сценарий `database_connection.php`, который вы можете применять снова и снова.

В действительности вам нужен еще один сценарий, показывающий информацию о пользователе. Тогда можно будет просто переместить пользователей из сценария `create_user.php`, создающего пользователей, в этот новый сценарий и дать ему возможность определить, что нужно сделать с точки зрения ответа. Поэтому оставим пока `create_user.php` в незавершенном виде, чтобы позже можно было вернуться к нему и исправить код.

Покажите мне пользователя

Итак, задача состоит в следующем: вам нужна страница, показывающая информацию о пользователе в том виде, который имеет смысл для этого пользователя. Поэтому данная страница собирается извлекать информацию из таблицы `users`, но она не является формой. Здесь нет необходимости (по крайней мере пока) делать что-либо кроме отображения информации.

Теперь можно углубиться непосредственно в PHP, но основная работа не будет связана с кодом, она будет направлена на создание качественной страницы профиля пользователя. Поэтому немного приберегите стремление к освоению PHP на потом и начните с HTML.

Большинство веб-серверов настроено на прием запросов на файл, имя которого оканчивается на `.php`, и на создание HTML-вывода, который передается пользовательскому браузеру. Поэтому вы можете создать HTML, поместить его в файл с окончанием `.php`, а затем приступить к добавлению настоящего PHP, когда подоспеет время. Ваш веб-сервер отправит HTML, содержащийся в этом файле, запросившему его веб-браузеру, и ваши пользователи (или вы сами) увидят HTML-вывод.

Макетирование страницы профиля пользователя

На рис. 6.5 показана страница профиля, имеющая довольно внушительный вид. На ней приведена основная контактная информация пользователя, а также некоторые полезные добавления: краткая биография и фотография.

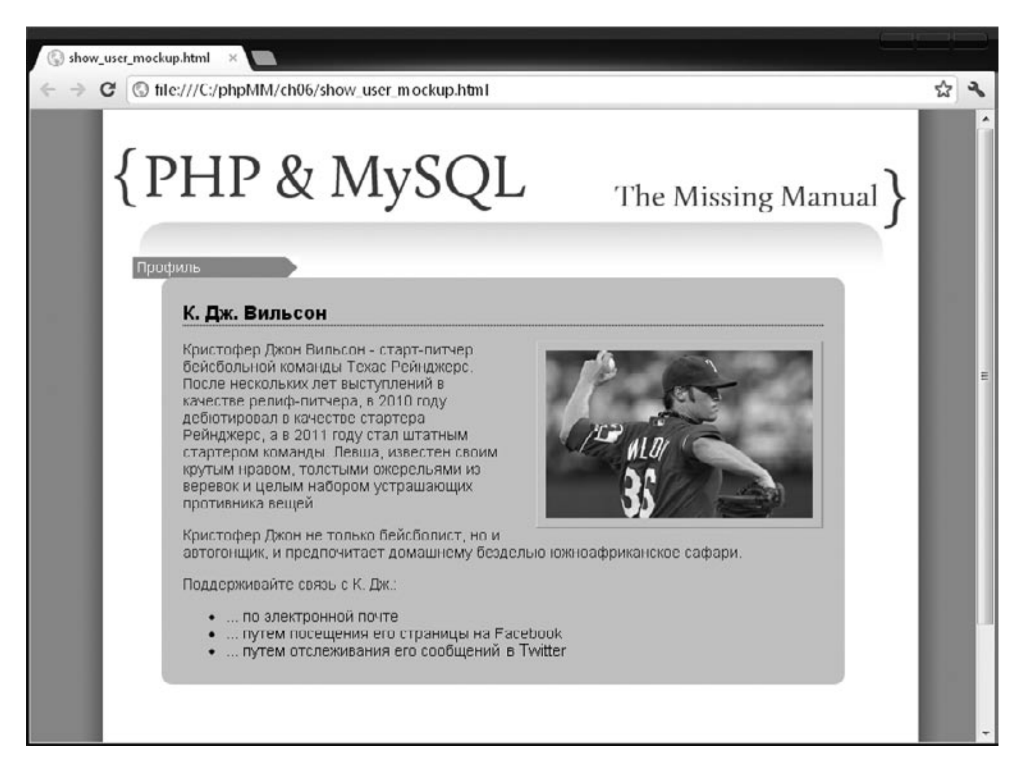


Рис. 6.5. Страница профиля

А вот как выглядит HTML для этой страницы (с CSS все получается не таким уж и сложным):

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Профиль</div>

  <div id="content">
    <div class="user_profile">
      <h1>К. Дж. Вильсон</h1>
      <p>
        <img src="images/cj_wilson.jpg" class="user_pic" />
      Кристофер Джон Вильсон - старт-питчер бейсбольной
      команды Техас Рейнджерс. После нескольких лет выступлений
      в качестве релиф-питчера, в 2010 году дебютировал в
      качестве стартера Рейнджерс, а в 2011 году стал штатным
      стартером команды. Левша, известен своим крутым характером,
      толстыми ожерельями из веревок и целым набором устрашающих
```

```
      противника вещей. </p>
     <p>Кристофер Джон не только бейсболист, но и автогонщик,
      и предпочитает домашнему безделью южноафриканское сафари.
     </p>
      <p class="contact_info">Поддерживайте связь с К. Дж.:</p>
      <ul>
        <li>...
          <a href="wilson@texasrangers.com">по электронной почте</a></li>
        <li>... путем
          <a href="http://www.facebook.com/pages/CJ-Wilson/127083957307281">
           посещения его страницы на Facebook</a></li>
        <li>... путем <a href="http://www.twitter.com/str8edgeracer">
        отслеживания его сообщений в Twitter </a></li>
      </ul>
    </div>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

ПРИМЕЧАНИЕ

Биография и фото здесь являются новыми элементами, которых еще нет в таблице users. Они даны для улучшения внешнего вида страницы профиля. Просто имя и несколько ссылок на электронную почту и Twitter смотрелись бы слишком скучно.

Но переживать не нужно, совсем скоро и биография, и фото для профиля будут добавлены в базу данных, и тогда ваше приложение сможет вывести именно эту страницу.

Продолжим работу и представим (или сделаем) эту страницу не с текстом-заместителем, а с переменными на месте этого текста. Итак, там, где должно быть имя пользователя, просто представим `$first_name`, а затем представим, где должны находиться `$last_name`, `$email` и т. д. Результат вполне очевиден:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Профиль</div>

  <div id="content">
    <div class="user_profile">
      <h1>$first_name $last_name</h1>
      <p><img src="$user_image" class="user_pic" />
        $bio</p>
      <p class="contact_info">Поддерживайте связь с $first_name:</p>
      <ul>
        <li>...
          <a href="$email">по электронной почте</a></li>
        <li>... путем
```

```
        <a href="$facebook_url">посещения его страницы на
        Facebook</a></li>
      <li>... путем <a href="$twitter_url">отслеживания его сообщений
        в Twitter</a></li>
    </ul>
  </div>
 </div>
 <div id="footer"></div>
 </body>
</html>
```

ВНИМАНИЕ

Этот пример хорош для обдумывания, но он не является приемлемым кодом HTML или PHP. Поэтому не пытайтесь просматривать этот код в браузере, вы не получите ничего полезного. Он, конечно же, не будет работать в качестве сценария PHP. Но вы должны понять, что почти все на странице на самом деле представляет информацию, имеющуюся в базе данных, и все представлено в весьма удобном для пользователя формате.

Этот пример показывает, насколько конструктивной является идея сконцентрироваться сначала на HTML, не углубляясь сразу же непосредственно в PHP. Приступая к созданию своей страницы, вы часто задумываетесь о том, что вам действительно нужны, например, биография (`$bio`) и фотография (`$user_image`), а не о том, что на данный момент имеется в вашей таблице `users`.

Итак, после этого простого макетирования выяснились важные обстоятельства.

- В таблице `users` не хватает важной информации. Нам нужны биография, представляющая собой длинный текстовый фрагмент, и способ загрузки фотографии пользователя.
- После обновления страницы нужно будет обновить файл `create_user.html` и форму `create_user.php`, чтобы дать возможность пользователям ввести эту информацию, а затем созранить новую информацию в базе данных.
- И наконец, самое существенное: внесение этих изменений позволит создать страницу профиля пользователя, имеющую вполне привлекательный вид.

Итак, с чего начать? Обычно в центре всего стоит база данных, поэтому нужно обновить таблицу `users`.

Изменение структуры таблицы с помощью ALTER

Пользователям не хватает двух информационных составляющих: биографии и фотографии. Добавление фотографии рассмотрим позже. Это потребует немного большего объема работы. Вы всегда можете поместить вместо нее указатель места заполнения, а потом вернуться к решению этой задачи. Ну а что касается биографии, то здесь все просто.

Сначала нужно изменить структуру таблицы базы данных, добавив еще один столбец. Именно этим занимается SQL-команда `ALTER`:

```
ALTER TABLE users
        ADD bio varchar(1000);
```

Команда так же проста, как выглядит. SQL-команде дается таблица для внесения изменений (проведения операции ALTER), а затем указывается, какие именно изменения следует внести. В данном случае вам нужно добавить столбец, поэтому используется ключевое слово ADD, которому дается имя и тип нового столбца.

Разумеется, здесь есть и тонкость: может ли пользователь оставлять поле биографии пустым или столбец bio должен иметь свойство NOT NULL? (Наверное, ничего страшного, если поле будет пустым, поэтому свойство NOT NULL не понадобилось.) Как информация для нового пользователя попадет в этот столбец? (Вам нужно обновить HTML-веб-форму create_user и сценарий, заставляющий работать базу данных. Это будет нашим следующим шагом.) Можно ли вносить изменения в структуру таблицы, когда только заблагорассудится? Конечно, ведь без такой возможности база данных просто ничего бы не стоила.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Что происходит со старыми строками в таблице при добавлении нового столбца?

При всей простоте добавления столбца к таблице базы данных с помощью команды ALTER и относительно несложном обновлении форм с целью предоставления пользователям возможности поместить информацию в такие столбцы (и демонстрации результатов, если у вас уже есть сценарий show_user) остается все же один сложный вопрос: как работать со старыми данными, у которых неожиданно появился новый столбец?

Возьмем таблицу users и представим, что в ней не одна-две записи, а тысячи пользователей за последние пять лет. Теперь после описанного выше изменения каждый из этих пользователей приобрел зияющую пустоту — место для своей биографии. Большинство баз данных оставляет этот столбец пустым, следовательно, при каждой попытке извлечь что-нибудь из нового столбца bio вы будете получать NULL.

Получение значения NULL в данном случае не такая уж большая проблема. Наверное, биографию пользователя можно будет назвать «нововведением», сопроводив все это пресс-релизом, и преподнести упущение в виде совершенно новой версии, повышающей качество ресурса и удобство для всех потенциальных пользователей, предпочитающих иметь дело с биографией. Существующие пользователи могут войти в систему и добавить свою биографию, чего они не могли сделать раньше.

Добавление нового требуемого столбца к таблице базы данных может вызвать определенные затруднения. Представьте владельцев веб-сайта, решивших, что использование адреса электронной почты в качестве имени пользователя не самая лучшая идея. Возможно, они изменят свои таблицы, добавят столбец username, но должны будут придать ему свойство NOT NULL. Зачем, в конце концов, имя пользователя без требования его обязательного присутствия? Разве есть пользователи без имен?

В результате будет огромное количество строк, утративших требуемые данные. Что нужно сделать? Можно просто заблокировать пользователей с незаполненным столбцом username и к следующей их попытке попасть на сайт создать механизм, заставляющий их выбрать имя пользователя. Это вполне обычное явление в наши небезопасные для Интернета времена. А что если пребывание всех этих строк в ненадлежащем состоянии, пока пользователь не войдет в систему, не отвечает нормам надежности?

Если проблема в этом, а так чаще всего и бывает, может понадобиться вставить какой-нибудь заместитель данных в таблицу, например NEEDS_USERNAME (требуется имя пользователя), и просто попросить пользователя при его возвращении на сайт проверить, является ли данное значение принадлежащим ему именем пользователя. Это, конечно, не самое элегантное решение, но оно позволит сохранить данные в правильном состоянии.

В конечном счете при использовании ALTER возникает весьма серьезная проблема: вам потенциально на какой-то период времени приходится мириться с неверными данными или же довольствоваться «фальшивыми» данными, чтобы сохранить работоспособность системы, хотя вы знаете, что в конечном итоге данные не могут оставаться в таком виде. Ни одно из решений нельзя признать удачным, поэтому нужно просто выбрать меньшее из двух зол. (Или вообще сделать что-нибудь другое.)

ПРИМЕЧАНИЕ

Работа с изображениями будет рассмотрена чуть позже. Еще многое предстоит рассказать о работе с изображениями и о том, где их хранить. Поэтому вы можете запланировать наличие изображения и оставить для него место на веб-странице и в сценарии.

Создание сценария: первый проход

Итак, располагая новым столбцом bio в таблице users и HTML-макетом, вы готовы заняться кодом PHP. Создадим новый сценарий, который назовем show_user.php (показать пользователя). Этот сценарий неплохо впишется в один ряд со сценарием create_user.php (создать пользователя). Можно представить, что позже для завершения всей линейки будут добавлены такие сценарии, как delete_user.php (удалить пользователя) и update_user.php (обновить пользователя).

Для начала в этом сценарии вообще не понадобится какой-либо код PHP. Вместо него поместим в сценарий код HTML. Затем, как уже делалось ранее, вы можете заменить все места, где должны быть данные, полученные из базы данных, именами PHP-переменных. В конечном итоге должен получиться следующий код:

```
<html>
 <head>
  <link href="../../css/phpMM.css" rel="stylesheet" type="text/css" />
```

```
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Профиль</div>

  <div id="content">
    <div class="user_profile">
      <h1>$first_name $last_name</h1>
      <p><img src="$user_image" class="user_pic" />
        $bio</p>
      <p class="contact_info">Поддерживайте связь с $first_name:</p>
      <ul>
       <li>...
          <a href="$email">по электронной почте</a></li>
        <li>... путем
          <a href="$facebook_url">посещения его страницы на
          Facebook</a></li>
        <li>... путем <a href="$twitter_url">отслеживания его сообщений
          в Twitter</a></li>
      </ul>
    </div>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

ПРИМЕЧАНИЕ

Не забудьте сохранить этот файл с расширением PHP в своем каталоге scripts/. Это может быть каталог ch06/scripts/, если вы придерживаетесь структуры книги, или просто каталог scripts/ вашего веб-сайта, если вы помещаете весь свой PHP-код со всех глав в одно место.

Теперь рассмотрим вполне очевидные странности.

- Где здесь код PHP? Раз нет <?php или ?>, значит, нет и кода.
- Переменные, начинающиеся со знака доллара ($), определенно относятся к PHP, а не к HTML. HTML-страница не будет знать, что с ними делать.
- Где осуществляется взаимодействие с базой данных? Здесь нет кода SQL, нет команды выборки из базы данных SELECT или каких-либо других команд.
- Какой пользователь должен быть загружен? Откуда сценарий знает, какого пользователя загружать?

Все эти вопросы действительно стоит задать. И если вы на некоторые из них сможете ответить, значит, вы уже стали разбираться в серьезных вопросах, касающихся PHP и веб-программирования.

Сначала разберемся в том, куда делись теги <?php и ?>. Вы можете дать файлу расширение PHP и не поместить в него ничего, кроме кода HTML. Наберите URL сценария в своем браузере и посмотрите, что из этого получится. Примерная картина показана на рис. 6.6.

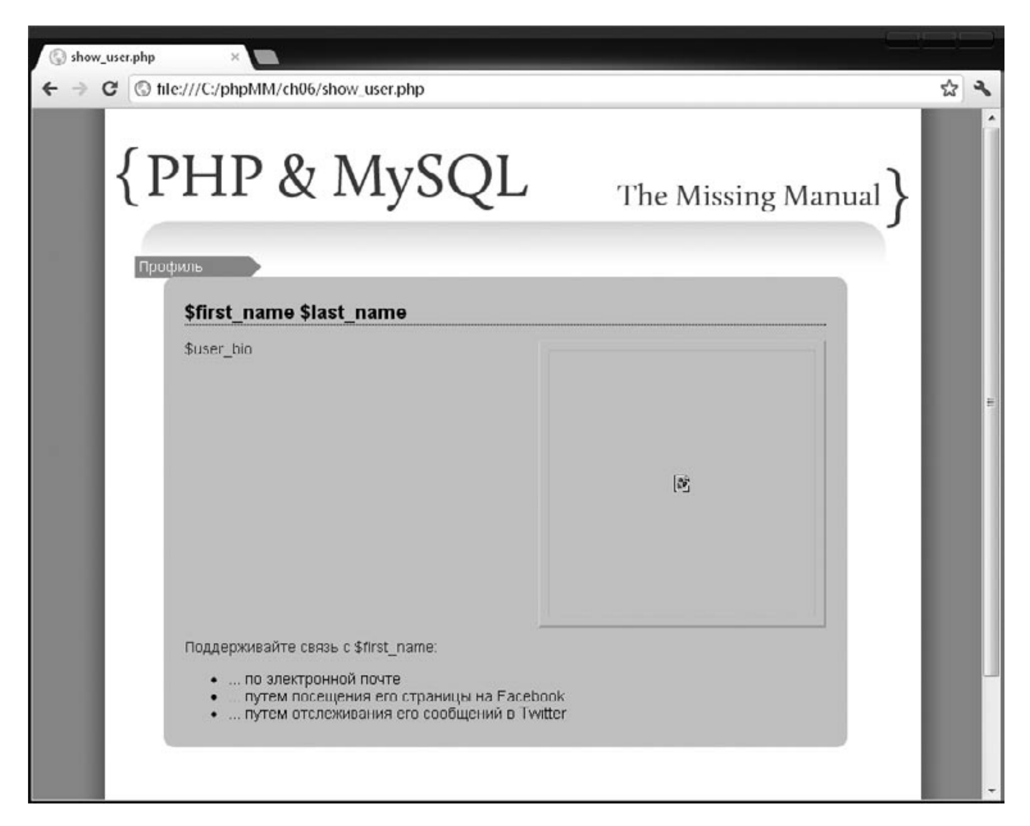


Рис. 6.6. Результат выполнения кода

СОВЕТ

Еще одной неплохой технологией тестирования будет начало работы с кода HTML. На данной стадии вам не нужны какие-либо настоящие значения, поэтому можно сконцентрироваться на внешнем виде страницы. После добавления кода PHP вам будет проще сконцентрироваться на взаимодействии с вашей базой данных и на форматировании фактически существующих строк и значений в ваших переменных. Когда перед взаимодействием с базой данных создается некий прототип, можно обеспечить нужный внешний вид, а затем скорректировать его еще раз, как только для каждой переменной будет вставлено ее реальное значение.

ПРИМЕЧАНИЕ

Если у вас получилась страница без стилевого оформления, то вполне возможно, что вам придется обновить элемент link в разделе head страницы. После перемещения сценария в каталог scripts/ CSS может оказаться в другом месте относительно HTML-страницы в корневом веб-каталоге или в каталоге примеров ch06/.

На данный момент в файле `show_user.php` нет ничего, кроме HTML, поэтому ваш веб-сервер предоставляет этот HTML веб-браузеру пользователя. В результате получается вполне приличная веб-страница. Разумеется, на ней требуется провести еще целый ряд доработок, например касательно тех имен переменных, которые выводятся в виде текста.

Все это не составит особого труда. Можно просто окружить каждую переменную тегами <?php и ?>, что скажет системе: «Это нужно воспринимать как фрагмент PHP». Затем следует добавить инструкцию echo, поскольку вам нужно вывести значение этой переменной:

```
<html>
 <head>
  <link href="../../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Профиль</div>
  <div id="content">
    <div class="user_profile">
      <h1><?php echo "{$first_name} {$last_name}"; ?></h1>
      <p><img src="<?php echo $user_image; ?>" class="user_pic" />
        <?php echo $bio; ?></p>
      <p class="contact_info">
        Поддерживайте связь с <?php echo $first_name; ?>:
      </p>
      <ul>
        <li>...
          <a href="<?php echo $email; ?>">по электронной почте</a></li>
        <li>... путем
          <a href="<?php echo $facebook_url; ?>">посещения его страницы
           на Facebook</a></li>
        <li>... путем <a href="<?php echo $twitter_url; ?>">отслеживания
           его сообщений в Twitter</a></li>
      </ul>
    </div>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

И здесь возникает вполне очевидная проблема: у переменных, начинающихся со знака доллара ($), нет значений. Вы их еще не определили, и при попытке обращения к этим переменным на данной странице будут получены весьма странные результаты (рис. 6.7). (При каждом запуске код PHP с помощью команды echo выводит значение несуществующей переменной. PHP поступает здесь несколько неопределенно и просто ничего не выводит. В этом проявляется его косноязычие, но именно так PHP и делает: выводит «никакую строку», которая просто является пустым местом.) Но вы все же медленно продвигаетесь к получению полноценного сценария.

Самая большая проблема заключается в том, что в действительности вам не известно, работает ли этот код. Есть ли в нем опечатки? Есть ли какие-нибудь проблемы с тем имеющимся у вас минимальным кодом PHP? Если вы не уверены, что все будет работать должным образом, то перейти к коду для работы с вашей базой данных будет непросто, даже если в базе данных будут правильные значения.

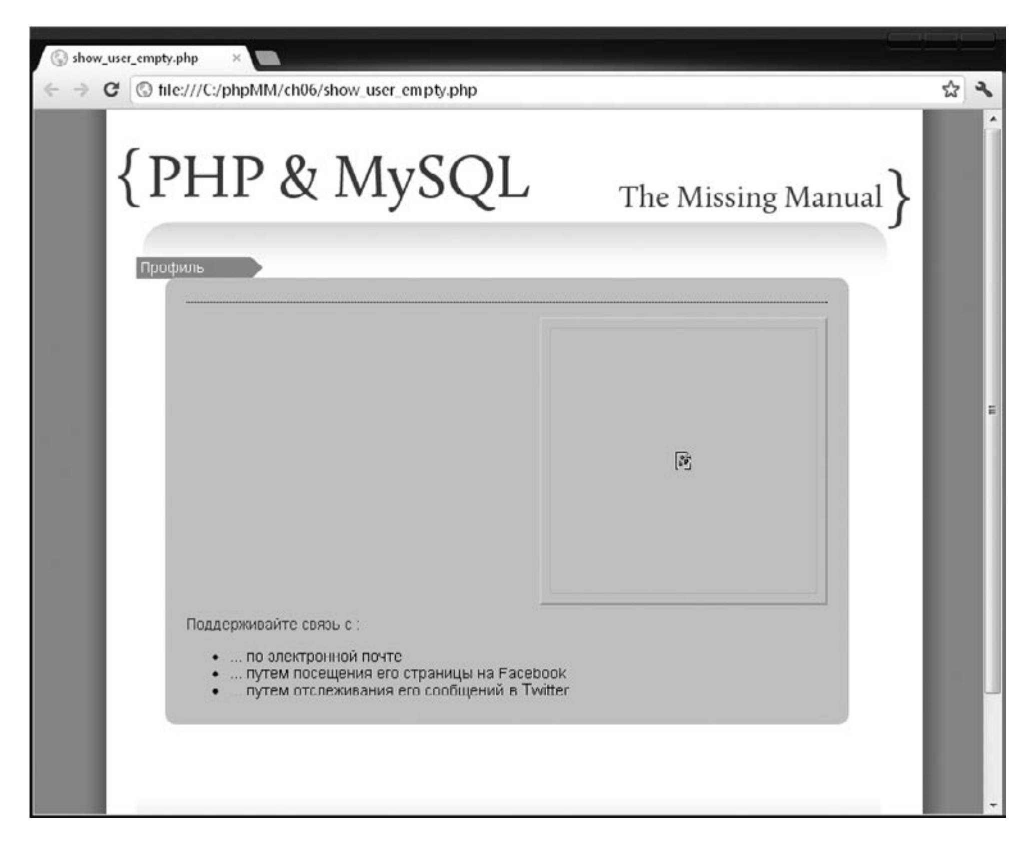


Рис. 6.7. Полученная веб-страница

Один из несложных способов тестирования перед продолжением разработки заключается в простом создании небольшого раздела PHP-кода перед HTML. В <?php-разделе нужно присвоить каждой переменной то значение, которое будет получено из базы данных:

```
<?php
$first_name = "К. Дж.";
$last_name = "Вильсон";
$user_image = "/еще/не/создано.jpg";
$bio = "Кристофер Джон Вильсон - старт-питчер бейсбольной
      команды Техас Рейнджерс. После нескольких лет выступлений
      в качестве релиф-питчера, в 2010 году дебютировал в
      качестве стартера Рейнджерс, а в 2011 году стал штатным
      стартером команды. Левша, известен своим крутым характером,
      толстыми ожерельями из веревок и целым набором устрашающих
      противника вещей. </p>
     <p>Кристофер Джон не только бейсболист, но и автогонщик, и
      предпочитает домашнему безделью южноафриканское сафари.";
$email = "wilson@texasrangers.com";
$facebook_url = "http://www.facebook.com/pages/CJ-Wilson/127083957307281";
```

```
$twitter_url = «http://www.twitter.com/str8edgeracer»;
?>

<html>
  <!-- Весь код HTML и встроенный в него код PHP -->
</html>
```

Теперь вы можете посетить свою страницу и посмотреть на кое-какие полезные результаты, показанные на рис. 6.8. Вы можете увидеть, что код действительно работает, и теперь остается только дать переменным, начинающимся со знака доллара ($), реальные значения, а затем определить, сведения о каком пользователе нужно искать в первую очередь.



Рис. 6.8. Страница по-прежнему является макетом. Тем не менее это хороший способ разработки полноценного сценария: постепенно на каждом шагу убеждаться в независимой работоспособности кода от всего остального, что будет добавлено позже

Выбор пользователя из базы данных с помощью инструкции SELECT

У вас есть переменные и есть код HTML. Теперь нужно получить сведения о вашем пользователе. Это совсем нетрудно, учитывая, что вы уже несколько раз применяли инструкцию SELECT:

```
SELECT *
  FROM users;
```

Фактически эту команду уже можно запустить в отношении вашей базы данных и получить все имеющиеся строки:

```
+---------+------------+-----------+----------------+-------------
-----------------------------------------+---------------+------+
| user_id | first_name | last_name | email                   |
facebook_url                                                 |
twitter_handle  | bio  |
+---------+------------+-----------+----------------+-------------
-----------------------------------------+---------------+------+
|      1 | К. Дж.     | Вильсон  | wilson@texasrangers.com | http://www.facebook.com/
pages/CJ-Wilson/127083957307281 |
@str8edgeracer  | NULL |
+---------+------------+-----------+----------------+-------------------
-------------------------------------+---------------+------+
1 row in set (0.03 sec)
```

ПРИМЕЧАНИЕ

Этот вывод намеренно оставлен в таком беспорядочном состоянии, поскольку вы, наверное, увидите примерно то же самое. Инструкция SELECT выводит из таблицы абсолютно все столбцы, которые не могут поместиться в обычном окне командной строки, не говоря уже о ширине данной страницы.

В данном случае у нас имеется всего один пользователь. Поэтому, получив о нем сведения, можно извлечь значения столбцов `first_name` и `last_name`, значение `email` и т. д. и поместить их в `$first_name`, `$last_name` и в остальные используемые вами переменные.

Но остается еще один важный вопрос: откуда вы знаете, о каком пользователе нужно извлекать сведения? Разумеется, в имеющемся выводе из таблицы всего один пользователь. А что будет, когда ваше новое приложение наберет популярность и у вас будут сотни, тысячи и даже сотни тысяч пользователей? Вам нужно выбрать только одного из этих пользователей для сценария `show_user.php`, чтобы вывести сведения о нем, но... как определить, какого именно пользователя нужно выбрать?

Чтобы понять это, подумайте, как люди будут добираться до сценария `show_user.php`. Вот несколько вариантов.

- Они будут переправлены на данную страницу после создания нового пользователя с помощью `create_user.html` и `create_user.php`.
- Они войдут в приложение и щелкнут на ссылке вроде **Мой профиль** или **Обновить сведения обо мне**.
- Они выберут конкретного пользователя из списка пользователей. Может быть, это будет список всех пользователей, находящихся в системе, или список всех их друзей, или всех пользователей, которых они читают в Facebook или в Twitter.

Что общего во всех этих ситуациях? Никто не переходит к `show_user.php` непосредственно, путем ввода URL. В каждом конкретном случае пользователи вашего приложения выбирают пользователя, создают пользователя или входят как пользователи, а затем какая-нибудь ссылка приводит их к сценарию `show_user.php`.

В результате в каждой логичной ситуации ваш код отправляет пользователя к show_user.php, и следовательно, ваш код контролирует ситуацию. Если понадобится, скажем, отправить show_user.php какую-то информацию, то вы в состоянии это сделать. Все, что вы делаете, заключается в отправке уникального ID того пользователя, сведения о котором сценарий show_user.php должен загрузить из базы данных, а затем вывести на экран.

Итак, еще раз обратимся к тому же сценарию.

- Пользователь создается с помощью сценария create_user.php, и ID этого нового пользователя передается наряду с информацией о пользователе вашего приложения сценарию show_user.php.
- Щелчок на ссылке Мой профиль или Обновить сведения обо мне приводит к передаче сценарию show_user.php текущего вошедшего в программу пользователя.
- Выбор пользователя из списка, независимо от содержимого этого списка, приводит к переходу по ссылке на show_user.php и к одновременной передаче этому сценарию ID выбранного пользователя.

В каждом случае ваш сценарий show_user.php должен получить переданный ему ID, затем найти пользователя по переданному ID и наконец показать этого пользователя.

Преимущество данного решения не только в возможности его реализации, но и в наличии у вас контроля над всеми способами возможных подходов пользователей к сценарию show_user.php. Дополнительное преимущество состоит в том, что ID показываемого пользователя можно передать как часть запроса, а ранее вы уже извлекали составляющие из запроса, используя переменную $_REQUEST.

Теперь добавьте к show_user.php следующую строку:

```
<?php

$user_id = $_REQUEST['user_id'];

// Код присваивания значений переменным страницы
?>

<html>
  <!-- Весь код HTML и встроенный в него код PHP -->
</html>
```

Здесь нет ничего нового. Единственное изменение по сравнению с тем, что было сделано раньше, заключается в извлечении параметра запроса с новым именем: user_id.

Теперь наконец-то можно инструкции SELECT добавить условие WHERE:

```
SELECT *
  FROM users
 WHERE user_id = $user_id;
```

Вам уже попадались условия WHERE (например, в подразделе «Использование базы данных с помощью команды USE» раздела «SQL — язык для разговора с базами данных» главы 3). Они действуют в соответствии с вашими возможными

ожиданиями: сужают набор результатов на основе совпадения или каких-нибудь других ограничений. В данном случае говорится следующее: «Мне нужно все (*) из таблицы users, но только для тех записей (строк), у которых столбец user_id имеет такое же значение, как и у переменной $user_id».

Итак, если у вашего пробного пользователя столбец user_id содержит 1 и значение переменной $user_id равно 1, вы получите данные этого пробного пользователя. Если в таблице нет строк со столбцом user_id, содержащим 1, то инструкция SELECT ничего не вернет. А вот что действительно интересно: поле user_id с помощью ключевых слов PRIMARY KEY было объявлено первичным ключом (см. подраздел «ID и первичные ключи — хорошие компаньоны» раздела «Проектирование таблиц базы данных» данной главы), следовательно, у вас не может быть более одного возвращенного результата. Поэтому вам не нужно смотреть, сколько значений было возвращено, или делать что-либо особенное, чтобы справляться с одной или с несколькими строками. Вы либо не получите в ответ вообще ничего, поскольку не было совпадений, либо получите всего одну строку.

Если все это собрать воедино, можно сделать к show_user несколько действительно важных добавлений.

PHP:

```
<?php

require '../../scripts/database_connection.php';

// Получение ID показываемого пользователя
$user_id = $_REQUEST['user_id'];

// Создание инструкции SELECT
$select_query = "SELECT * FROM users WHERE user_id = " . $user_id;

// Запуск запроса
$result = mysql_query($select_query);

// Присваивание значений переменным
?>

<html>
  <!-- Весь код HTML и встроенный в него код PHP -->
</html>
```

Теперь с помощью этого кода происходит подключение к вашей базе данных, создается инструкция SELECT из переданного параметра запроса user_id и запускается запрос. Остается только создать абсолютно новый фрагмент сценария: пройтись по конкретному результату запроса и получить информацию из этого результата.

Извлечение значений из результата SQL-запроса

Вы получили переменную $result, но что она собой представляет? Возможно, вы помните, что это ресурс, переменная специального типа, в которой содержится ссылка на дополнительную информацию. Следовательно, это ресурс можно пере-

дать другим PHP-функциям и использовать его для получения дополнительной информации.

В случае с применением SELECT-запроса вам нужны все строки, возвращенные этим запросом, а затем для каждой строки необходимо получить различные значения. Именно для этого и можно задействовать ресурс, следовательно, вы собираетесь завершить создание сценария show_user.php и приступить к принятию запросов.

Сначала убедимся, что у переменной $result есть значение. Это равноценно проверке, не имеет ли $result значение false, которое возвращается при возникновении проблем с SQL:

```
// Выдача запроса
$result = mysql_query($select_query);

if ($result) {
  // Получение возвращенных запросом строк с помощью $result
} else {
  die("Ошибка обнаружения пользователя с ID {$user_id}");
}
```

Эта инструкция if также (в некоторой степени) обрабатывает ошибки. Если переменная $result имеет значение false, значит, что-то пошло не так, что, по-видимому, означает отсутствие пользователя, поиск которого велся с помощью $user_id, или что при поиске этого пользователя возникла проблема. Этот код в данном виде не создает слишком привлекательного формата сообщения об ошибке и фактически дает слишком мало информации о причине, вызвавшей ошибку. Но ничего страшного, очень скоро в обработку ошибок будут внесены нужные изменения, поэтому данная инструкция if является вполне нормальным временным решением.

Теперь вам нужна новая PHP-функция: mysql_fetch_array. Она забирает ресурс у предыдущего запущенного SQL-запроса. Это именно то, что есть в переменной $result:

```
if ($result) {
  $row = mysql_fetch_array($result);

  // Разбиение строки на имеющиеся в ней разные поля и присваивание значений
  // переменным
} else {
  die("Ошибка обнаружения пользователя с ID {$user_id}");
}
```

И здесь возникают некоторые странности. Обратите внимание на то, что сценарий, использующий показанный выше код, сохраняет результат вызова функции mysql_fetch_array в переменной $row. Следовательно, mysql_fetch_array возвращает из SQL-запроса одну строку, и это так на самом деле и есть. Но имя функции указывает на другое: на то, что она возвращает массив[1]. Так что же именно она должна возвращать? И то и другое. Функция mysql_fetch_array возвращает массив, но она возвращает массив для отдельной строки запроса, связанной с переданным ей результатом.

[1] Array — с англ. массив. — *Примеч. ред.*

Итак, при вызове функции mysql_fetch_array($result) вы собираетесь получить отдельную строку результатов, но при данном способе возвращения строки она возвращается в виде массива.

ПРИМЕЧАНИЕ

Здесь нет ничего удивительного. Вы, конечно же, можете получить любую строку результатов, возвращенных запросом, а не только самую первую строку. Для этого вы просто должны вызывать функцию mysql_fetch_array снова и снова, и она будет возвращать следующую строку результатов. В конечном итоге функция mysql_fetch_array вернет false, показывая тем самым, что результаты кончились.

Вскоре вы сами станете использовать mysql_fetch_array подобным образом, и все станет на свои места. А пока следует понять, что при каждом вызове этой функции вы получаете одну строку результатов (или значение false, если строк больше не осталось) и эта строка является массивом значений.

Массивы для вас не проблема, поэтому возвращение массива в переменной $row является хорошей новостью. В сущности, содержимое $row просто похоже на другой, уже известный вам массив $_REQUEST (см. раздел «Переменная $_REQUEST» главы 2). И так же, как при работе с $_REQUEST, вы имеете дело не просто со списком значений, а со значениями, связанными с ключами на основе имен.

По этой причине, когда запрос приходил с параметром по имени "first_name", вы извлекали значение для этого параметра с помощью выражения $_REQUEST['first_name']. Тот же самый принцип применяется к переменной $row. Ей можно дать имя столбца, возвращенного вашим SQL-запросом, и вы получите значение этого столбца в рассматриваемой строке.

Итак, получив переменную $row, вы можете получить значения всех нужных вам столбцов и присвоить эти значения переменным:

```
// Запуск запроса
$result = mysql_query($select_query);
if ($result) {
  $row = mysql_fetch_array($result);
  $first_name     = $row['first_name'];
  $last_name      = $row['last_name'];
  $bio      = $row['bio'];
  $email          = $row['email'];
  $facebook_url   = $row['facebook_url'];
  $twitter_handle = $row['twitter_handle'];

  // Превращение $twitter_handle в URL
  $twitter_url = "http://www.twitter.com/" .
                 substr($twitter_handle, $position + 1);

  // Для последующего добавления
  $user_image = "/еще/не/созданное_изображение.jpg";
} else {
  die("Ошибка обнаружения пользователя с ID {$user_id}");
}
```

ПРИМЕЧАНИЕ

В конце этой инструкции if вы должны добавить ранее использовавшийся код, создающий URL для идентификатора в Twitter. Это тот же самый код, который применялся для создания подобного URL в пункте «Удаление лишних пробелов с помощью функции trim()» подраздела «Обрезка и замена текста» раздела «Изменение текста» главы 2, хотя тогда вы еще не получали идентификатор в Twitter из базы данных.

Нужно также добавить код, присваивающий переменной $user_image значение-пустышку в расчете на последующее возвращение к этому коду и ее замену на настоящее изображение пользователя. При отсутствии фото можно также применять типовое изображение:

```
$user_image = "../../images/missing_user.png";
```

Если хотите сделать это прямо сейчас, то образец подобного изображения можно найти в загружаемых примерах для данной главы.

Итак, вы получили полностью работоспособный сценарий! Фактически весь код, за исключением определения порядка использования ресурса `$result` с функцией `mysql_fetch_array`, уже вполне отработан и не создаст вам абсолютно никаких проблем.

Получение ID пользователя сценарием show_user.php

Настало время получить вашему сценарию ID пользователя, чтобы он мог применять этот ID для поиска пользователя, получения информации о нем и вывода этой информации. Но перед тем, как вы начнете разбираться со всеми другими своими сценариями, неплохо было бы убедиться в работоспособности сценария `show_user.php`. Разве вам захочется тратить уйму времени на другие сценарии, если есть еще не решенные проблемы в сценарии `show_user.php`? Это будет бесполезной тратой времени.

К счастью, существует весьма простой способ тестирования вашего сценария. Массив `$_REQUEST` содержит всю информацию, переданную в сценарий из связанного с ним запроса, включая дополнительную информацию, переданную через сам URL запроса. Теперь в первую очередь нужно запомнить, что ни способ получения информации для `show_user.php`, ни даже доступ к `show_user.php` в данном случае не является обычным. Вместо них сценарий будет получать информацию из таких сценариев, как `create_user.php` или, может быть, от кнопки типа Мой профиль.

Но теперь мы просто проводим тестирование. По этой причине почему бы сразу не перейти на страницу с URL, подобным следующему: yellowtagmedia.com/phpMM/ch06/scripts/show_user.php? При доступности данного адреса можно снабдить этот сценарий требуемыми данными с помощью параметров запроса в самом URL. Эти параметры можно просто добавить к этому URL после символа `?`. Формат имеет следующую основу:

```
[протокол]://[имя_домена]/[местонахождение_файла]?[параметры_запроса]
```

Можно использовать такой URL: mysite.com/scripts/show_user.php?first_name=Lance. После чего можно будет взять значение `$_REQUEST['first_name']` и получить `"Lance"`. Можно также составить пакет из нескольких параметров, отделяя их друг от друга

символом &. Следовательно, можно пойти еще дальше и использовать такой URL: mysite.com/scripts/show_user.php?first_name=Lance&last_name=McCollum.

ПРИМЕЧАНИЕ

С формальной точки зрения имя файла (show_user.php) представляет собой путь, а информация за ним (?first_name=Lance&last_name=McCollum) является строкой запроса.

Что делать дальше, вы можете разобраться самостоятельно. Добавьте ID того пользователя, который был создан заранее (или одного из пользователей, если в таблицу вставлено более одного пользователя), и проверьте работу `show_user.php` с URL, похожим на yellowtagmedia.com/phpMM/ch06/scripts/show_user.php?user_id=1.

В ответ вы должны получить изображение, похожее на то, что показано на рис. 6.9, что станет проверкой всей работы по созданию запроса SQL и кода `show_user.php`.



Рис. 6.9. Результирующая страница

ВНИМАНИЕ

Параметры запроса, как и код PHP, чувствительны к регистру букв. Поэтому запрос $_REQUEST['user_id'] не будет соответствовать запросу с параметром USER_ID или user_Id. Нужно следить за точным совпадением букв в верхнем и нижнем регистрах.

На данный момент сделано практически все возможное для того, чтобы сценарий `show_user.php` заработал должным образом. В нем еще не хватает некоторой

информации, такой как фотографии пользователя и его биографии, но с фотографией можно разобраться позже, а с биографией нужно разобраться с помощью обновления сценария `create_user.php`. Кроме того, настало время оставить в покое сценарий `show_user.php` и снова обратиться к сценарию, отправляющему ему пользователей.

ПРИМЕЧАНИЕ

Наверное, потребуется еще одно действие: необходимо будет с помощью команды INSERT вручную вставить пользователя с биографией в вашу таблицу users, а затем снова посмотреть на работу сценария show_user.php. Это может понадобиться прямо сейчас, чтобы как следует убедиться, что сценарий show_user.php отвечает всем вашим желаниям. Эти функциональные возможности будут протестированы сразу же после обновления create_user.php, поскольку тестировать-то там особо нечего.

ЗАМЕТКИ О ПРОЕКТИРОВАНИИ

Разумно ли создавать каталог scripts/?

Хранение всех сценариев в каталоге (или с технической точки зрения в подкаталоге) по имени scripts/ является обычаем, который по большому счету возник во времена таких более старых языков программирования, как Perl и CGI (эта аббревиатура означает Common Gateway Interface, то есть общий шлюзовой интерфейс, предназначенный для вызова внешних программ, таких как сценарии на стороне сервера). В те времена было четкое разделение между программами на стороне клиента, или представлениями, и программами на стороне сервера. Поэтому сценарии фактически никогда не занимались тем, что приводило к непосредственному отображению веб-страницы, они были просто программами, вызываемыми другими процессами.

Но PHP по сути стер грани между сценарием и просматриваемой страницей. Сценарий `show_user.php` содержит фактически больше HTML, чем PHP, и он будет прост для работы пользователя, который попал в него напрямую. Иными словами, PHP — это не только способ написания сценариев, к которым ваши формы закулисно отправляют свои данные. Можно назвать немало случаев, когда пользователи щелкают на ссылках на PHP-страницу, а не на страницу HTML или даже набирают URL PHP-сценария в адресных строках своих браузеров.

Существуют особенно популярные программные средства, в которых весь HTML управляется по сути с помощью PHP. Например, распространенная система управления блогами и содержимым Wordpress (wordpress.org и wordpress.com) построена на PHP. В этой системе главной страницей вашего сайта является `index.php`, а не `index.html`.

Итак, есть ли в таком случае смысл в использовании каталога scripts/? По большому счету смысла нет. Пока внешний вид страницы и ее действия отвечают ожиданиям ваших пользователей, им все равно, откуда она получена, из HTML-файла или из PHP-сценария. И добавление каталога scripts/ просто увеличивает количество необходимых пользователям сведений о вашей системе, но не делает что-либо более понятным.

Поэтому начиная с главы 7 это изменение, касающееся хранения сценариев в каталоге scripts/, вступит в силу. Было бы неплохо, чтобы вы чувствовали разницу между тем, что будете делать в качестве создателя веб-страницы с HTML, CSS и JavaScript, и тем, что будете делать, пользуясь своими новым навыками в области программирования на PHP. Но сейчас, когда вы уже вышли за рамки передачи форм коду PHP, настало время продолжить стирание граней и позволить многим вашим PHP-сценариям находиться рядом с вашим HTML.

Перенаправление и повторное обращение к сценарию, создающему новых пользователей

Все внесенные до сих пор изменения, конечно, весьма существенны, но недостаточны. Есть новый столбец bio, но пользователям негде вводить в него информацию при регистрации. Нужно создать такой сценарий create_user.php, который работал бы с этой информацией, когда она поступает из вашей формы регистрации. А затем забирал данные пользователя из формы регистрации в сценарий show_user.php и передавал вместе с ними ID только что созданного пользователя. Казалось бы, весьма солидный объем работы, но с вашими знаниями справиться со всем этим будет нетрудно.

Обновление формы регистрации

Первое изменение будет одним из самых простых. Откройте исходный файл create_user.html и добавьте в форму новое поле, позволяющее вашим пользователям вводить биографию. Оставьте в ней достаточно места, ведь вы же видели, как много информации люди в наши дни пишут о себе в Facebook?

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Регистрация пользователя</div>

  <div id="content">

    <h1>Вступайте в наш виртуальный клуб</h1>
    <p>Пожалуйста, введите ниже свои данные для связи в Интернете:</p>
    <form action="scripts/create_user.php" method="POST">
      <fieldset>
        <label for="first_name">Имя:</label>
        <input type="text" name="first_name" size="20" /><br />
        <label for="last_name">Фамилия:</label>
        <input type="text" name="last_name" size="20" /><br />
```

```
        <label for="email">Адрес электронной почты:</label>
        <input type="text" name="email" size="50" /><br />
        <label for="facebook_url">URL-адрес в Facebook:</label>
        <input type="text" name="facebook_url" size="50" /><br />
        <label for="twitter_handle">Идентификатор в Twitter:</label>
        <input type="text" name="twitter_handle" size="20" /><br />
        <label for="bio">Биография:</label>
        <textarea name="bio" cols="40" rows="10"></textarea>
      </fieldset>
      <br />
      <fieldset class="center">
        <input type="submit" value="Вступить в клуб" />
        <input type="reset" value="Очистить и начать все сначала" />
      </fieldset>
    </form>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

Можно также заодно позволить своим пользователям выбрать изображение, применяемое для их профиля. Пока создавать какой-нибудь код для обработки изображения в `create_user.php` не нужно, но вскоре этим все равно придется заняться. А если внести изменения в форму прямо сейчас, то не придется возвращаться к файлу `create_user.html`, когда вы будете готовы добавлять изображения.

```
<html>
 <!-- раздел заголовка -->

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Регистрация пользователя</div>

  <div id="content">
    <h1>Вступайте в наш виртуальный клуб</h1>
    <p>Пожалуйста, введите ниже свои данные для связи в Интернете:</p>
    <form action="scripts/create_user.php" method="POST"
         enctype="multipart/form-data">
    <fieldset>
     <!-- Другие поля -->

     <label for="user_pic">Отправка изображения:</label>
     <input type="file" name="user_pic" size="30" />

    </fieldset>
    <!-- Кнопки для отправки формы и ее перезапуска -->

 </body>
</html>
```

Если раньше вам не приходилось работать с отправкой файлов на сервер, то этот код HTML может показаться слегка странным. Вам следует немного изменить

тег form, поскольку теперь вы имеете дело с отправкой файла на сервер с машины пользователя. Поэтому добавьте новый атрибут enctype со значением multipart/form-data. Это добавление позволяет любым сценариям, получающим данные, которые введены в эту форму, ожидать не просто значения из полей ввода вроде имен файлов. Подобные формы отправляют также данные, связанные с этими полями. В данном случае это будет файл, выбранный пользователем для отправки на сервер.

Затем добавляется новое поле ввода типа file, позволяющее пользователю просматривать содержимое его жесткого диска, выбирать файл и отправлять его на сервер. Этот код является почти шаблоном, но он применяется не только для этого. Такие изменения нужно вносить при каждом предоставлении вашим пользователям возможности отправлять файл на сервер.

ПРИМЕЧАНИЕ

Если продолжать рассуждения на эту тему, то важным вопросом будет следующий: «Где хранить это изображение?» Вам нужно разрешить пользователю отправить изображение на сервер, а это потребует от ваших сценариев и кода работать с ним. Но что с ним делать, сохранять в файловой системе сервера и ссылаться на него, используя поле в вашей таблице users? Или же хранить это изображение в базе данных? Вокруг этого идут горячие споры, но для принятия решения вам придется изучить еще несколько глав.

Сохраните изменения и попробуйте вызвать форму в браузере. Обновленная форма должна иметь вид, показанный на рис. 6.10. Таким образом, после добавле-

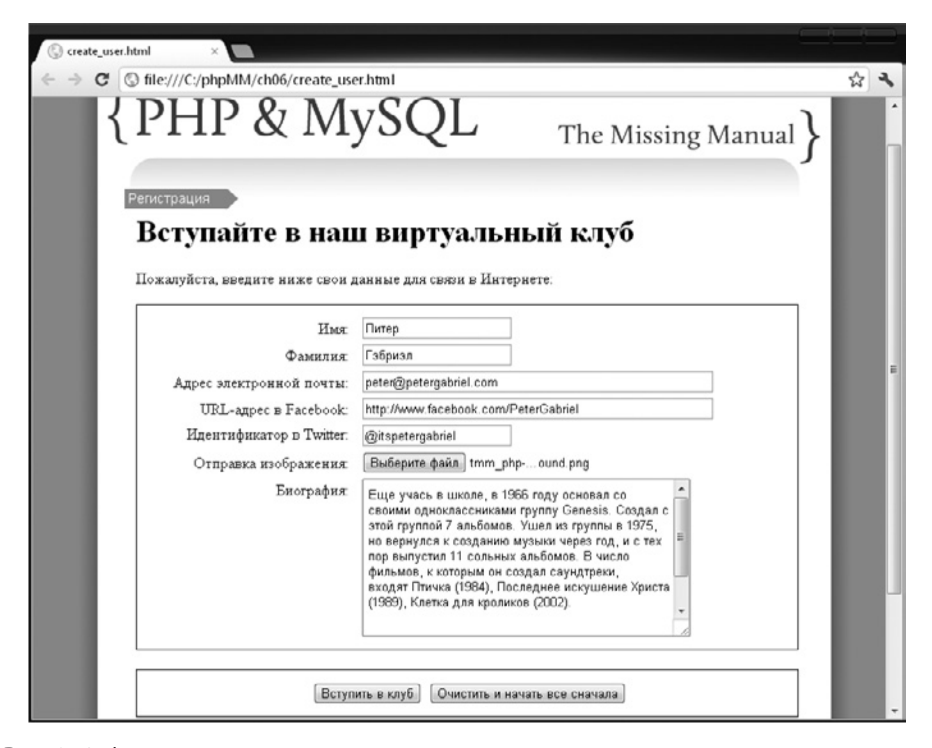


Рис. 6.10. Форма теперь отправляет не только название, которое появляется в каждом поле формы, но и все, что связано с этим названием

ния нового поля ввода или изменения формы для отправки составных данных внешний вид формы почти не изменился. Однако форма отправляет теперь не только название, которое появляется в каждом поле формы, но и все, что связано с этим названием, например файл, который был выбран после того, как пользователь щелкнул на кнопке Выберите файл.

Можете продолжить работу и заполнить поля по своему усмотрению, но без внесения изменений в сценарий create_user.php, вы получите пользователя без биографии.

Обновление сценария создания пользователя

Вы уже, наверное, догадались, что нужно сделать со сценарием create_user.php. Необходимо просто взять новую переменную запроса bio, добавить ее к инструкции INSERT, и все будет в порядке:

```
<?php

require '../../scripts/database_connection.php';

$first_name = trim($_REQUEST['first_name']);
$last_name = trim($_REQUEST['last_name']);
$email = trim($_REQUEST['email']);
$bio = trim($_REQUEST['bio']);

// И далее другие переменные запроса...

$insert_sql =
  "INSERT INTO users (first_name, last_name, email, bio,
                      facebook_url, twitter_handle)
" .
  "VALUES ('{$first_name}', '{$last_name}',
                      '{$email}', '{$bio}' " .
                      "'{$facebook_url}', '{$twitter_handle}');";

// Вставка пользователя в базу данных
mysql_query($insert_sql);
?>
```

Вот и все. Теперь вы можете отправлять новую форму и получать новый столбец — bio со значениями, успешно попадающими в вашу базу данных.

ВНИМАНИЕ

Не забудьте перед тем, как испытать код в действии, запустить инструкцию ALTER TABLE, добавляющую столбец bio в таблицу users.

Можете попробовать, как этот работает, заполнив форму create_user.html и щелкнув на кнопке отправки формы. Затем попробуйте запустить следующую инструкцию SELECT:

```
SELECT first_name, last_name, bio
  FROM users;
```

Результаты должны сказать сами за себя:

```
| first_name | last_name | bio
| Питер      | Гэбриэл   |Еще учась в школе, в 1966 году основал со своими
одноклассниками группу Genesis. Создал с этой группой 7 альбомов. Ушел из группы
в 1975, но вернулся к созданию музыки через год, и с тех пор выпустил 11 сольных
альбомов. В число фильмов, к которым он создал саундтреки, входят Птичка (1984),
Последнее искушение Христа (1989), Клетка для кроликов (2002).
```

ПРИМЕЧАНИЕ

Ваш результат будет отличаться, поскольку он будет основан на той биографии, которую вы ввели для своего учебного пользователя. Вы можете также увидеть старых пользователей. В данном случае запись пользователя C. J. Wilson в поле bio имеет значение NULL, поскольку этот пользователь был создан до того, как был создан столбец bio.

Затем нужно будет перенаправить вашего пользователя на сценарий show_user.php, а затем каким-то образом получить в этот сценарий ID только что созданного пользователя.

Первое требование выполнить нетрудно:

```
<?php

// Все, что уже было сделано

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php");
exit();
?>
```

Функция header буквально отправляет простой HTTP-заголовок (HTTP — HyperText Transfer Protocol, то есть протокол передачи гипертекста) браузеру вашего пользователя. Подробности, касающиеся HTTP, вам знать не нужно, достаточно понимать, что это язык веб-трафика. (Об использовании протокола HTTP говорит префикс http://, который ставится впереди большинства URL в адресном поле браузера.) Таким образом, вы непосредственно управляете местонахождением страницы вашего пользователя.

В данном случае местонахождение изменено с текущего на новое: на сценарий show_user.php. Но чтобы все работало, следует учесть несколько моментов.

- Функцию header нужно вызвать перед любым другим выводом в сценарии. Вы не можете выводить до этого вызова что-нибудь с помощью команды echo, ни тег <html>, ни что-либо еще. Функция header должна следовать перед любым выводом, иначе возникнут проблемы.
- Ссылка на местонахождение должна быть в виде относительного или абсолютного URL. Следовательно, можно указать в качестве места, например, http://www.google.com или ../../scripts/database_connection.php. А в данном случае сценарий в том же самом каталоге, что и текущий — show_user.php.

Несмотря на свою простоту, эти правила играют важную роль. Им нужно следовать, иначе функция header потерпит неудачу.

Теперь остается только разобраться с ID пользователя. Вы уже знаете, что функция mysql_query, выполняющая вашу инструкцию INSERT, возвращает ресурс, а не ID пользователя. И вся идея здесь заключается не в выборе пользователя из базы данных с помощью инструкции SELECT, а в том, чтобы оставить того же пользователя для show_user.php.

Чтобы show_user.php получил ID пользователя, требуется средство, отступающее на один шаг от ваших текущих знаний PHP: невероятно полезная PHP-функция по имени mysql_insert_id. Такие функции попадаются довольно редко, если только не задать поиск, скажем, функции для получения ID последней добавленной с помощью инструкции INSERT строки в таблицу базы данных, имеющую столбец автоприращения AUTO_INCREMENT.

Это определение подходит для функции mysql_insert_id. Она создана именно для того, что вам нужно: для получения ID без всяких дополнительных инструкций SELECT или приложения каких-нибудь других усилий.

Хотя функции mysql_insert_id можно передать ресурс, она автоматически использует тот ресурс, который был открыт последним, что вполне подойдет для данной ситуации. Просто добавьте ее вызов после инструкции INSERT, вызванной с помощью функции myql_query, и она автоматически получит ссылку на ресурс, возвращенный после этого вызова.

Она вернет ID нужного вам пользователя. Вы можете даже поставить ее в конец URL, точно так же, как при наборе вашего URL вручную:

```
<?php

// Все, что уже было сделано

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . mysql_insert_id());
?>
```

Вот и все. Добавьте этот код к сценарию create_user.php и можете посмотреть, как все работает.

ПРИМЕЧАНИЕ

У вас может появиться желание использовать следующий код:

("Location: show_user.php?user_id={mysql_insert_id()}");

К сожалению, он не будет работать. Интерпретатор PHP отлично справляется со вставкой значений переменных вместо их имен в фигурных скобках, как в следующем случае:

("Location: show_user.php?user_id={$user_id}");

Но он не будет делать то же самое с вызовами функций.

Зайдите в форму ввода данных при создании пользователей и заполните ее. Отправьте данные, и вы будете вознаграждены не только результатами работы

сценария `create_user.php`, но и отображением данных с помощью сценария `show_user.php`, загружающего данные только что созданного пользователя. На рис. 6.11 показан результат получения указания на пользователя.



Рис. 6.11. Информация о созданном пользователе

Представьте себе все, что здесь происходит. Вы набираете текст, щелкаете кнопкой мыши, и форма отправляется вашему сценарию на сервере. Этот сценарий вставляет данные в базу данных. Затем он заставляет браузер пользователя обратиться к другому сценарию, который запрашивает у базы данных все о конкретном пользователе. И наконец, ваш пользователь получает возможность увидеть все это буквально через несколько мгновений. Это уже существенно выходит за рамки простого использования HTML, CSS и JavaScript. Добро пожаловать в веб-программирование!

Усовершенствование кода с помощью регулярных выражений (в очередной раз)

Ваш сценарий *почти* доведен до совершенства. Какие же проблемы остались нерешенными? Из-за того, что текст отображается в виде одного общего блока, выводимая информация сильно проигрывает в привлекательности. Но есть ли какой-нибудь способ дополнительного форматирования этого текста?

В примере, показанном на рис. 6.11, пользователь нажимал между строками клавишу Enter, но это никак не отразилось на коде HTML. Нужен простой и быстрый способ замены этих нажатий клавиши Enter HTML-тегами <p></p>.

Необходимо найти способ поиска определенных символов, а точнее, весьма специфичных символов, и замены их вполне конкретными другими символами. Вы в курсе, как это сделать. Ведь эти нажатия клавиши Enter показываются в виде символов \r или \n или некой комбинации из этих двух символов и для их поиска и замены можно использовать регулярные выражения.

Обновите show_user.php для преобразования нажатий клавиши Enter в HTML-теги <p>, воспользовавшись для этого функцией preg_replace:

```
<?php

// Код подключения к базе данных

// Выбор нужного пользователя с помощью инструкции SELECT

if ($result) {
  $row = mysql_fetch_array($result);
  $first_name     = $row['first_name'];
  $last_name      = $row['last_name'];
  $bio            = preg_replace("/[\r\n]+/", "</p><p>", $row['bio']);
  $email          = $row['email'];
  $facebook_url   = $row['facebook_url'];
  $twitter_handle = $row['twitter_handle'];

  // Создание URL Twitter
}
?>

// HTML-вывод
```

ПРИМЕЧАНИЕ

Здесь нужно использовать [\r\n]+, а не [\r\n]*. Символ * будет задавать признак отсутствия в тексте символов, указанных в квадратных скобках, и их комбинаций, и вы получите совершенно ненужную комбинацию </p><p> между каждым символом в биографии пользователя. А символ + гарантирует, что \r или \n (или обе комбинации) появляются хотя бы однажды перед тем, как заменяются комбинацией тегов </p><p>.

Здесь вы можете оценить эффективность применения регулярных выражений. Они избавляют вас от циклических проходов и поисковых операций, и вам не нужно определять, какие именно символы были введены пользователем на разных платформах — \r, \n или \r\n. Вы просто подключаете нужное регулярное выражение, и больше ничего не нужно.

Если собрать все воедино, должна получиться следующая версия show_user.php:

```
<?php

require '../../scripts/app_config.php';
```

```
require '../../scripts/database_connection.php';

// Получение ID показываемого пользователя
$user_id = $_REQUEST['user_id'];

// Создание инструкции SELECT
$select_query = "SELECT * FROM users WHERE user_id = " . $user_id;

// Запуск запроса
$result = mysql_query($select_query);

if ($result) {
  $row = mysql_fetch_array($result);
  $first_name     = $row['first_name'];
  $last_name      = $row['last_name'];
  $bio            = preg_replace("/[\r\n]+/", "</p><p>", $row['bio']);
  $email          = $row['email'];
  $facebook_url   = $row['facebook_url'];
  $twitter_handle = $row['twitter_handle'];
  // Превращение $twitter_handle в URL
  $twitter_url = "http://www.twitter.com/" .
                 substr($twitter_handle, $position + 1);

  // Для последующего добавления
  $user_image = "../../images/missing_user.png";
} else {
  die("Ошибка обнаружения пользователя с ID {$user_id}");
}
?>

<html>
 <head>
  <link href="../../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Профиль</div>

  <div id="content">
    <div class="user_profile">
      <h1><?php echo "{$first_name} {$last_name}"; ?></h1>
      <p><img src="<?php echo $user_image; ?>" class="user_pic" />
        <?php echo $bio; ?></p>
      <p class="contact_info">
        Поддерживайте связь с <?php echo $first_name; ?>:
      </p>
      <ul>
        <li>... по электронной почте
          <a href="<?php echo $email; ?>"><?php echo $email; ?></a></li>
        <li>... путем
```

```
          <a href="<?php echo $facebook_url; ?>"> посещения его страницы
          на Facebook</a></li>
        <li>... путем <a href="<?php echo $twitter_url; ?>">отслеживания его
             сообщений в Twitter </a></li>
      </ul>
    </div>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

Попробуйте это в деле, и, как показано на рис. 6.12, в конечном итоге вы увидите не только информацию о пользователе, но и красиво отформатированную биографию.

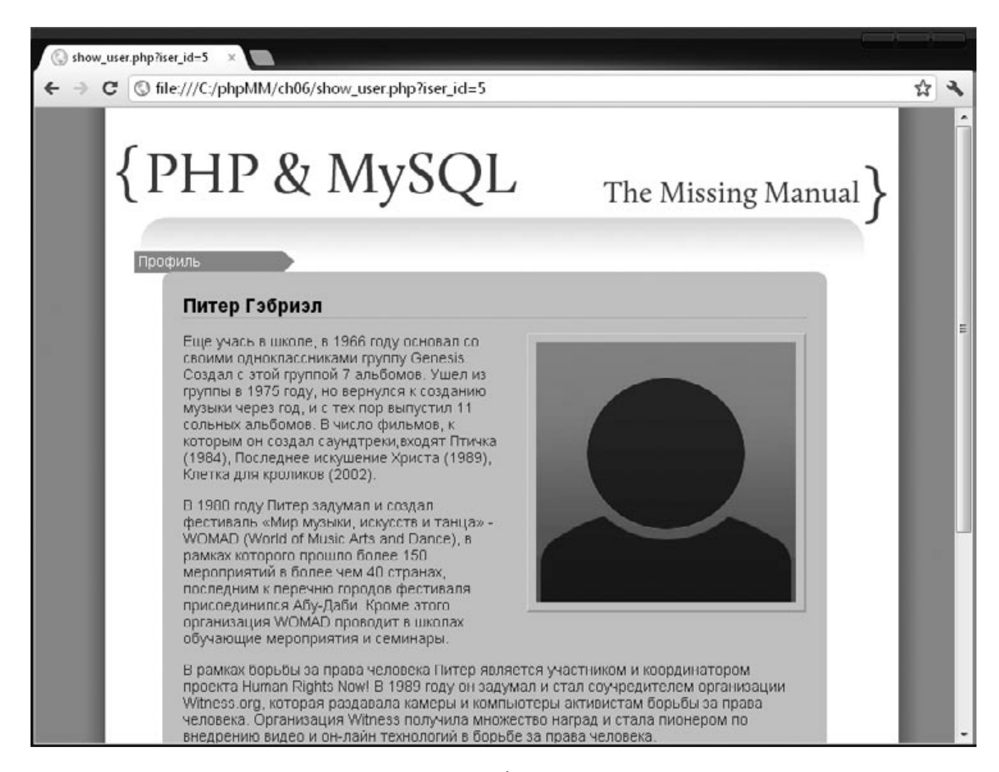


Рис. 6.12. Биография пользователя

Таким образом, это еще один классический пример проявления деликатности по отношению к своим пользователям. Правильно ли с точки зрения функциональности извлекать биографию пользователя из базы данных и показывать ее? Конечно, правильно. Правильно ли с той же точки зрения не вставлять в биографию код HTML при ее сохранении? Разумеется, правильно. Но когда биография выводится на экран, пользователям все равно, что находится в вашей базе данных. Их волнует только то, как она выглядит.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Должны ли совпадать имена полей, переменных и столбцов таблицы?

Вы могли заметить связующую нить между именем поля в HTML в `create_user.html`, которая тянется к сценарию `create_user.php`, а также к другим сценариям вроде `show_user.php` и доходит до самой таблицы базы данных. Например, имя `first_name` остается неизменным в HTML, PHP и MySQL (а следовательно, и в SQL). Это не является каким-то требованием, вы можете назвать поле `firstName`, а переменную — `user_firstName` и вызвать столбец `first_name`. Пока поддерживается определенный порядок, весь ваш код будет работоспособен. Итак, совершенно необязательно, чтобы все ваши имена совпадали.

Но может быть лучше задать такой вопрос: «Должна ли существовать преемственность имен в HTML, PHP и MySQL?» Совпадающие имена придают системе постоянство. Вам не нужно лишний раз задумываться: «Теперь я знаю, как это называется в коде PHP, но какое имя было у соответствующего столбца в базе данных?»

Есть и обратная сторона медали: в различных языках программирования и системах управления базами данных существуют достаточно устоявшиеся соглашения относительно имен переменных. В Java желательно меньше пользоваться символом подчеркивания и больше применять разделение на основе заглавных букв. По этой причине в Java будет отдано предпочтение имени `firstName`, а не `first_name`. Те же правила действуют и в C++, а вот в PHP и в таких языках, как Ruby, предпочтение отдается не использованию разничного регистра букв, а символам подчеркивания. В SQL также прослеживается благосклонность к символам подчеркивания.

Все это сводится к условному эмпирическому правилу: по возможности быть последовательным, не перемешивая соглашения, действующие в тех языках, на которых ведется программирование. Ваш код будет проще читать, начиная от самых внешних HTML-страниц и заканчивая самыми внутренними таблицами базы данных. Поскольку PHP относится к языкам, где предпочтение отдается символам подчеркивания, применяйте эти символы и сохраняйте простоту и последовательность в различных частях своего приложения.

ЧАСТЬ 3

Переход от веб-страниц к веб-приложениям

Глава 7. Когда что-то не получается (но должно получаться)

Глава 8. Обработка изображений и решение более сложных задач

Глава 9. Двоичные объекты и загрузка изображений

Глава 10. Вывод списков, итерация и администрирование

7 Когда что-то не получается (но должно получаться)

Пока у вас получился функционально нарастающий набор сценариев. Есть несколько веб-страниц, которые взаимодействуют с этими сценариями, CSS для формирования стилей как для статических HTML-страниц, так и для HTML, создаваемого вашими сценариями, и вы даже можете позволить себе добавить ряд проверок на стороне клиента. Похоже, дела идут неплохо.

ПРИМЕЧАНИЕ

Возьмите себе за правило добавлять проверку приемлемости данных на стороне клиента.

Но где-то в глубине затаилось чудовище. Хотя вы периодически и добавляли в код инструкцию `die` или инструкцию условного перехода, чтобы убедиться, что запросы вернули строку данных, код пока рассчитан на идеального пользователя, который всегда набирает только то, что вы от него ждете. Никогда не вводит номер телефона в поле для адреса электронной почты или пробелы в поле URL-адреса Facebook, никогда в неподходящий момент не щелкает на кнопке браузера для возврата на предыдущую страницу и никогда не вводит информацию о себе в одну и ту же форму дважды, нетерпеливо щелкая на кнопке отправки данных формы вместо ожидания реакции своего не самого быстрого интернет-подключения.

Конечно, никто не совершенен. Реальность такова, что при работе с веб-приложениями, да и с любым типом программ, люди всегда умудряются внести разлад в работу ваших страниц, форм и сценариев, рассчитанных на идеальный вариант обращения с ними. Они предоставляют некачественную информацию, не заполняют обязательные поля и вообще вносят путаницу.

ПРИМЕЧАНИЕ

Еще раз подчеркну: особую ценность в решении данной проблемы играет именно тот JavaScript, который выполняется на стороне клиента. Множество проблем такого сорта может быть решено путем проверки пользовательской информации еще до ее отправки вашим сценариям.

Итак, что же делать? До сих пор выполнялось нечто подобное показанному на рис. 7.1. И пока это довольно примитивное сообщение об ошибке нас вполне устраивало. Вы были единственным пользователем своей системы и просто проверяли, что происходит, убеждаясь в правильности кода. Но это слишком скудный способ обработки ошибок в любых системах, стремящихся на просторы диких

прерий Интернета. Подобные сообщения об ошибках мало что дают пользователю. Суть их непонятна, они раскрывают информацию о вашей системе, чего делать не должны, и, что еще хуже, они выглядят просто уродливо! В Интернете внешний вид играет большую роль, но еще важнее единый стиль оформления. Возникшие ошибки должны объясняться как можно понятнее и в формате, не выбивающимся из общей картины вашего сайта.

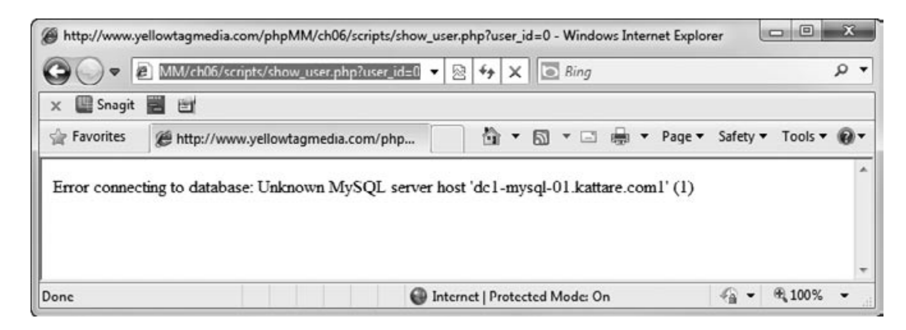


Рис. 7.1. Сообщение об ошибке

И это еще не все. Зайдите еще раз по URL сценария `show_user.php`, предоставив ID заведомо несуществующего пользователя. На рис. 7.2 показано, что ошибка



Рис. 7.2. Подобный пустой пользователь свидетельствует о наличии весьма неприятной проблемы, поскольку вообще не похоже, что возникла проблема

будет просто поглощена вашим сценарием. Вы получите пустой профиль пользователя, но все будет выглядеть так, будто ничего не случилось, даже притом, что сценарий получил неверный ID пользователя.

Итак, перед вами непочатый край работы. Но сначала нужно выяснить: что конкретно должно быть на странице ошибки.

Проектирование страниц ошибок

При создании страницы, на которой вы показывали пользователей, работа начиналась с HTML: делался макет простой страницы, а затем по мере необходимости добавлялся код PHP. Нет никаких причин отказываться от такого подхода и в данном случае, поскольку, по сути, будет выполняться нечто подобное. Вам нужна привлекательная страница для отображения ошибок, и перед углублением в PHP нужно получить приглядный внешний вид.

Итак, сначала необходимо создать HTML-страницу и назвать ее `show_error.html`. Начать можно с той же структуры, которая бралась за основу всех других страниц:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Страница ошибки</div>

  <div id="content">
    <h1>Страница ошибки</h1>
    <p>Ошибка</p>
  </div>

  <div id="footer"></div>

 </body>

</html>
```

Этот код создает пустую оболочку страницы ошибок (рис. 7.3), с которой можно приступить к работе.

Независимо от уровня вашего дизайнерского и программистского мастерства практически невозможно одновременно создать хороший дизайн и хороший код. Работая с макетом, а затем с кодом, вы можете последовательно сфокусироваться на выполнении каждой из задач. И как только вы на это настроитесь, практически всегда будет проще сначала смакетировать как минимум основной замысел внешнего интерфейса, а затем углубиться в код.

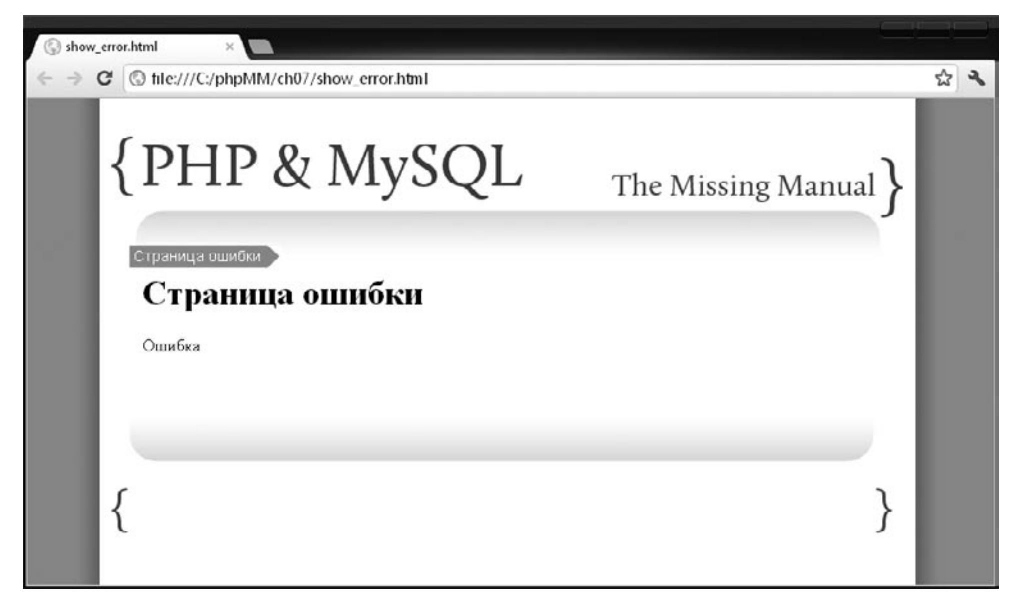


Рис. 7.3. Оболочка страницы ошибок

Что должны видеть пользователи?

Итак, первый вопрос: что именно должно попасть на страницу, что поможет вашим пользователям? Для ответа рассмотрим следующие два вопроса.

- Какая информация нужна вашему пользователю при возникновении ошибки?
- В какой манере должна быть преподнесена эта информация?

Сообщение пользователям о возникновении проблемы

Ответ на первый вопрос не должен вызывать сомнений. Произошел сбой, и вашему пользователю необходимы объяснения. Но даже здесь есть определенный нюанс. Нужно ли выводить сообщение об ошибке в том виде, в котором его может выдать MySQL одному из ваших сценариев?

```
#1054 - Unknown column 'firstname' in 'field list'
```

Наверное, в большинстве случаев нет. Если только ваш пользователь не MySQL- или PHP-программист, это сообщение будет абсолютно бесполезным. Его нужно перевести на обыкновенный человеческий язык:

```
К сожалению, найти пользователя в таким именем не удалось.
```

Это намного понятнее, но содержит слишком широко толкуемую информацию, создавая пользователю напрасную причину для беспокойства: «Почему они не могут найти мое имя? Неужели моя запись пропала? Есть ли мое имя в системе? Неужели моя запись была удалена? Что вообще происходит?!»

ПРИМЕЧАНИЕ

Вам подобная реакция кажется слишком драматизированной? Но она встречается нередко, особенно у тех пользователей, которые не доверяют Интернету свою личную информацию.

Тогда, может быть, это сообщение об ошибке должно быть достаточно понятным, но менее конкретизированным:

```
К сожалению, при обработке вашего запроса произошла ошибка.
```

Большинству людей так будет понятнее. Произошел сбой, что-то случилось. Ваша задача просто сообщить о наличии проблемы, не запугивая пользователя чудовищными подробностями.

Выбор манеры для сообщения об ошибке

Вы уже поняли, что ваш информационно подкованный пользователь действительно хочет знать о характере возникшей проблемы. Подробности, наверное, излишни и могут не развеять, а усилить тревогу. Но как насчет тона?

Звучит довольно эмоционально, но все же мы имеем дело с людьми, а значит, и с их эмоциями. Само по себе получение сообщения об ошибке уже является раздражающим фактором. Если ваше веб-приложение выдает ошибки, то на вас возлагается задача максимально сгладить стресс и чувство разочарования. Иначе люди перестанут пользоваться вашим приложением.

Это касается не только того, что говорится при возникновении проблемы, но и того, как это говорится. Строгое, безликое сообщение об ошибке выглядит менее утешительно по сравнению с сообщением в разговорной манере. Иногда можно

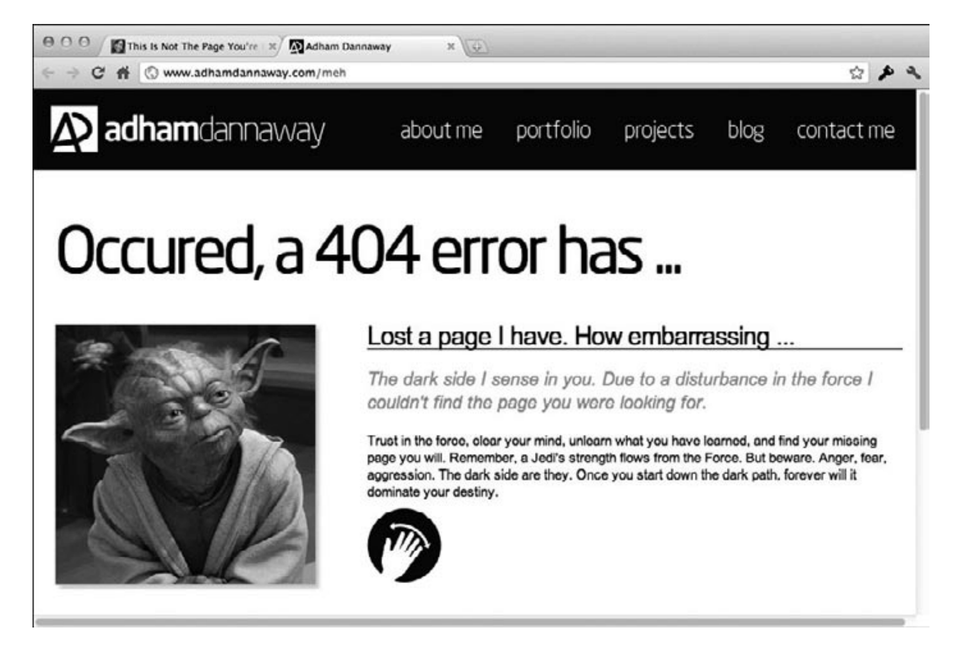


Рис. 7.4. Эта страница сообщает об ошибке, но в то же время здесь есть над чем посмеяться

даже придать сообщению юмористический оттенок. Посмотрите на рис. 7.4. Там показан юмористический подход к проблеме. (Об ошибке отсутствия страницы сообщается от имени одного из персонажей фильма «Звездные войны» в характерной для него манере путать порядок следования слов. — *Примеч. пер.*) Можно даже поспорить, что пользователь, попавший на такую страницу, касается она ошибки или нет, захочет вернуться на сайт.

Для вашего учебного сайта переходить на юмористический стиль подачи информации может быть и не нужно, но при этом следует хотя бы использовать разговорный стиль. Вполне достаточно будет уйти от таких строгих фраз, как «Ошибка 1282: Возникла исключительная ситуация».

Внесите, к примеру, немного разговорной манеры изложения в макет страницы ошибки и обратите внимание на то, как быстро повышается его привлекательность при возникновении неизбежной ошибки:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Извините</div>

  <div id="content">
    <h1>Нам очень жаль...</h1>
     <p><img src="../images/error.jpg" class="error" />...но произошел небольшой
сбой. Не волнуйтесь, мы в курсе происходящего и предпримем все
необходимые меры. Если же вы хотите связаться с нами и узнать подробности
произошедшего или вас что-нибудь беспокоит в сложившейся ситуации, пришлите
нам <a href="mailto:info@yellowtagmedia.com">сообщение</a>, и мы непременно
откликнемся.</p>
    <p>А сейчас, если вы желаете вернуться на страницу, ставшую причиной
проблемы, то можете <a href="javascript:history.go(-1);">щелкнуть здесь.</a>
Если возникнет такая же проблема, то вы можете вернуться на страницу чуть
позже. Уверены, что к тому времени мы во всем разберемся. Еще раз спасибо...
надеемся на ваше скорое возвращение. И еще раз извините за причиненные
неудобства.</p>

  </div>

  <div id="footer"></div>

 </body>

</html>
```

В этом сообщении говорится только лишь о том, что «Да, мы в курсе возникшей проблемы и работаем над ее устранением». Все остальное относится к манере преподнесения информации: повествовательный стиль, картинка, призванная разбавить

холодность страницы, и контактная ссылка для отправки сообщения по электронной почте, а также еще одна ссылка для повторного посещения проблемной страницы.

Посмотрите на рис. 7.5. Это сообщение имеет гораздо менее раздражающий вид, чем изображенное на рис. 7.3, а времени на его создание ушло ненамного больше.

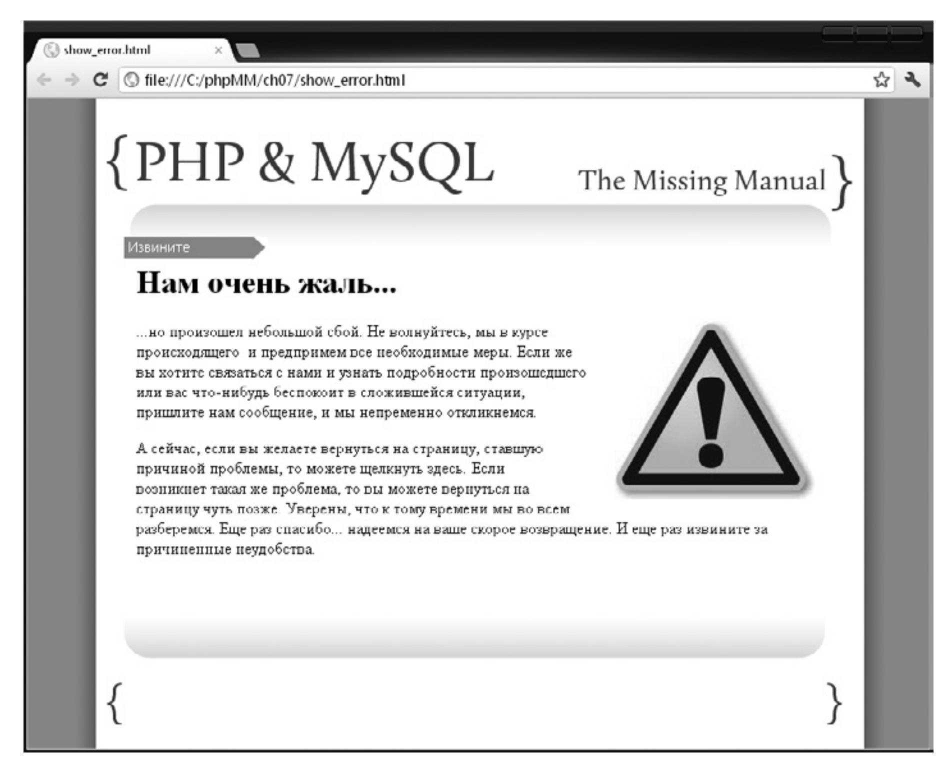


Рис. 7.5. Новое сообщение об ошибке

Эффективные страницы проведения проверок и сообщений об ошибках зачастую являются той гранью, которая разделяет случайных или неквалифицированных и высококвалифицированных программистов. Эти, казалось бы, незначительные детали поддерживают жизнеспособность систем и удовлетворенность пользователей, что в конечном итоге сохраняет репутацию и платежеспособность. Можете, конечно, пренебречь всем этим, но на свой собственный страх и риск!

Понятие о том, когда и сколько нужно говорить

Итак, теперь вы вполне квалифицированный PHP-программист, и у вас могут быть весьма светлые идеи о том, что может происходить на этой странице ошибки. Вы можете, скажем, взять пользовательскую информацию из базы данных и персонифицировать страницу. Вы можете создать новую таблицу с кодами ошибок и связанным с каждым из этих кодов полезным, удобочитаемым сообщением об ошибке. Затем, когда произойдет ошибка, вы можете найти ее код и вывести соответствующее сообщение об ошибке из базы данных.

Все эти идеи (и какие-нибудь еще, придуманные вами) могут пригодиться для создания весьма привлекательной страницы ошибки. Но они также требуют для своего создания довольно сложных приемов программирования. Нужно подключаться к базе данных и выполнять запросы. И при каждом создании запроса или подключении к базе данных появляется возможность для совершения еще одной ошибки! А куда деться вашим пользователям, когда ошибка будет на самой странице ошибки?

Нужно следовать главному правилу, которое требует, чтобы страницы ошибок как можно меньше зависели от программирования. Они не должны взаимодействовать с базами данных и не должны быть слишком вычурными. Если ваша страница ошибки сама может стать причиной ошибки, то у вас, мягко говоря, большие неприятности.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Все обещания — на вашей совести

На страницах ошибок легче всего дать обещание и не выполнить его. Если вы говорите пользователю, что вникаете в суть его проблемы, то лучше так и сделать. Если вы собираетесь предоставить контактный адрес электронной почты, проверьте его работоспособность (зачастую страницы ошибок устаревают и содержат недействующую контактную информацию) и наличие по этому адресу человека, способного решить проблему.

Если ваши пользователи полагают, что вы занимаетесь решением их вопроса и возвращаются через несколько часов на ту же страницу только для того, чтобы столкнуться с той же самой ошибкой, то вашему сайту не помогут никакие мудреные картинки и заумные слова. Более того, пользователей будет раздражать не столько сам сбой, сколько ваша очевидная ложь о работе по устранению проблемы.

Если вы только в начале своего пути или ограничены в ресурсах, лучше, наверное, будет просто сказать, что вы в курсе возникновения ошибки и ее устранение займет 24 или 36 часов или какой-нибудь другой период времени, за который вы можете поручиться. Можно также предоставить адрес электронной почты для решения срочных проблем, а затем следить за содержимым почтового ящика! Можно заранее придать почтовому сообщению формат, содержащий искомую строку темы, вроде «Срочно» или «Ошибка». Можно даже установить в вашей почтовой программе правило выделения сообщений по этой теме.

Чтобы вы ни делали, позаботьтесь о том, чтобы ваша реакция соответствовала обещаниям, которые даны на странице ошибки, или вам придется разбираться не только с проблемами программирования.

И еще один совет, который пригодится, когда вы начнете работать с крупными компаниями: никогда не позволяйте маркетинговым командам бесконтрольно составлять текст страниц ошибок! Не хочу навязывать стереотипы, но задача маркетинга заключается в продажах и продвижении товара, но в стремлении поднять репутацию компании они могут переоценить ваши возможности. Привлеките к корректировке вашей страницы ошибки специалиста, хорошо владеющего речью. Но в конечном итоге решать проблемы придется вам, и вы должны гарантировать выполнение всего, что обещано на страницах ошибок.

Поиск компромисса для страниц ошибок с помощью PHP

С одной стороны, вы стремитесь к предельной простоте своих страниц ошибок: немного текста, одно-два изображения и статический контент. Тогда ничего не может дать сбой, и ваши пользователи получают оределенную степень успокоенности и комфорта. А с другой стороны, страница ошибки на рис. 7.5 имеет слишком общий характер. На ней ни о чем особенном не говорится. Но было бы неплохо конкретизировать характер сбоя, может быть, следующим образом:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Извините</div>

  <div id="content">
    <h1> Нам очень жаль...</h1>
    <p><img src="../images/error.jpg" class="error" />...но произшел
небольшой сбой. Вероятно, <span class="error_message">введенное вами имя
пользователя не может быть найдено в нашей базе данных.</span></p>
    <p> Не волнуйтесь, мы в курсе происходящего и предпримем все необходимые
меры. Если же вы хотите связаться с нами и узнать подробности произошедшего
или вас что-нибудь беспокоит в сложившейся ситуации, пришлите нам
<a href="mailto:info@yellowtagmedia.com">сообщение</a>, и мы непременно
откликнемся.</p>
    <p> А сейчас, если вы желаете вернуться на страницу, ставшую причиной
проблемы, то можете <a href="javascript:history.go(-1);">щелкнуть здесь.</a>
Если возникнет такая же проблема, то вы можете вернуться на страницу чуть
позже. Уверены, что к тому времени мы во всем разберемся. Еще раз спасибо...
надеемся на ваше скорое возвращение. И еще раз извините за причиненные
неудобства.</p>

  </div>


  <div id="footer"></div>

 </body>

</html>
```

Результат, показанный на рис. 7.6, представляется неплохим компромиссом между страницей ошибки с сообщением общего характера и страницей, перенасыщенной специфичной для пользователя информацией, которая сама по себе ста-

новится источником ошибок. Из этого незначительного изменения по сравнению со страницей ошибки, показанной на рис. 7.5 и имеющей слишком общий характер, извлекаются два преимущества. Во-первых, теперь вы показываете пользователям сообщение, связанное с реальной проблемой. Эта небольшая персонализация дает им надежду, что вам известно, что нужно делать. А во-вторых, прикрепляя к этому сообщению CSS-класс, вы можете легко и просто изменить или обновить внешний вид данного сообщения.



Рис. 7.6. Обновленное сообщение об ошибке

Так как же можно получить такое персонализированное сообщение об ошибке и сохранить минимально возможный объем программирования?

Создание страницы ошибки с кодом PHP

Почти все в рассмотренном шаблоне страницы ошибки является простым кодом HTML. Все динамическое (иными словами, то, что изменялось бы от запроса к запросу) — это само сообщение об ошибке. Поэтому все превращается в относительно простое упражнение. Как обычно, можно начать с использования в коде HTML

PHP-переменной, предназначенной для хранения сообщения об ошибке, в расчете на то, что вы скоро вернетесь к этому месту сценария и присвоите этой переменной значение.

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Извините</div>

  <div id="content">
    <h1> Нам очень жаль...</h1>
    <p><img src="../images/error.jpg" class="error" />
      <?php echo $error_message; ?>
      <span></p>
    <p> Не волнуйтесь, мы в курсе происходящего и предпримем все необходимые
меры. Если же вы хотите связаться с нами и узнать подробности произошедшего
или вас что-нибудь беспокоит в сложившейся ситуации, пришлите нам
<a href="mailto:info@yellowtagmedia.com">сообщение</a>, и мы непременно
откликнемся.</p>
    <p> А сейчас, если вы желаете вернуться на страницу, ставшую причиной
проблемы, то можете <a href="javascript:history.go(-1);">щелкнуть здесь.</a>
Если возникнет такая же проблема, то вы можете вернуться на страницу чуть
позже. Уверены, что к тому времени мы во всем разберемся. Еще раз спасибо...
надеемся на ваше скорое возвращение. И еще раз извините за причиненные
неудобства.</p>

  </div>

  <div id="footer"></div>

</body>

</html>
```

Сохраните этот файл под именем show_error.php. И еще одна особенность: эта страница ошибки будет относиться ко всем сценариям и HTML-страницам. Поэтому не сохраняйте ее в каталоге главы 7. Вместо этого, чтобы упростить доступ к данному файлу, поместите ее в каталог scripts/ корневого каталога своего сайта.

ПРИМЕЧАНИЕ

Если вы хотите в точности следовать структуре данной книги, сохраните этот файл в каталоге phpMM/scripts/, где phpMM/ — корневой каталог интерактивных примеров.

Теперь вам нужно получить сообщение об ошибке. Как это сделать проще всего? Точнее, какой способ решения этой задачи наименее подвержен ошибкам? Наверное, метод с использованием параметров запроса и массива $_REQUEST.

```
<?php
  $error_message = $_REQUEST['error_message'];
?>

<html>
  <!-- Существующий HTML и PHP -->
</html>
```

Чем привлекателен данный подход? Он настолько элементарен, насколько только может быть простым программирование на PHP. Вы не используете никаких вычислений, а просто извлекаете значение из массива. Что еще лучше, этот массив создавали не вы, его для вас предоставил и даже заполнил интерпретатор PHP, используя информацию, приведенную в запросе к `show_error.php`.

Проверка принятого решения

С учетом всего вышесказанного проверьте работу страницы в браузере. Перейдите на URL вашего сценария и добавьте к нему параметр запроса. В этом URL можно воспользоваться следующими параметрами:

```
http://www.yellowtagmedia.com/phpMM/scripts/
show_error.php?error_message=There's%20been%20a%20problem
%20connecting%20to%20the%20database.
```

ПРИМЕЧАНИЕ

Весь этот URL должен набираться в адресной строке браузера. Если вы будете вводить пробелы, многие браузеры будут конвертировать их в безопасный для сети, но довольно странный эквивалент пробела %20. Тем самым браузер сообщает о том, что он вставляет пробел.

В результате получится весьма привлекательная страница ошибки, не потребовавшая для своего создания большого объема работы (рис. 7.7).

ПРИМЕЧАНИЕ

Одна из наиболее привлекательных сторон любого сценария, использующего параметры запроса и переменную $_REQUEST, заключается в возможности простой проверки таких сценариев с помощью командной строки. Нужно просто указать параметры в командной строке, отделив первый параметр от имени сценария символом ?, а затем отделив несколько параметров запроса друг от друга символами &.

Простота использования параметров запроса, которая заключается в том, что они представляют собой обычный текст, передаваемый от одной страницы или сценария другой странице или сценарию, и составляет всю прелесть сценария `show_error.php`. Здесь практически нет места для сбоев. Это именно та элегантность и простота, которые нужны для страницы ошибки.

Но нужно внести еще одну поправку: обратный слэш, который появляется перед одиночным апострофом, портит внешний вид сообщения. От него можно избавиться с помощью магии регулярного выражения. Замените все появления слэша пустой строкой:

```
$error_message = preg_replace_all("/\\\\/", '',
                                  $_REQUEST['error_message']);
```



Рис. 7.7. Проверка работы страницы

В PHP есть одна особенность, требующая указывать соответствие одному обратному слэшу с помощью четырех обратных слэшей. То есть, как ни странно, \\\\ соответствует \. Причина в том, что здесь происходит «борьба» с имеющимся в PHP механизмом нейтрализации специальных символов, в котором используется обратный слэш.

Ожидайте неожиданного

Все вроде бы неплохо, но опять-таки вы предполагаете, что происходит только то, что вам нужно. Фактически именно такие размышления приводят к игнорированию страниц ошибок. Но если вы в необходимости решать проблемы дошли до создания страницы ошибки, то лучше полагать, что проблемы могут также возникать, когда вы уже находитесь на странице ошибки.

К счастью, вы устранили возможность возникновения большинства таких проблем за счет простоты страницы. А что если параметр запроса `error_message` будет отсутствовать? Тогда вы получите страницу, показанную на рис. 7.8. «Вероятно...», а дальше ничего.

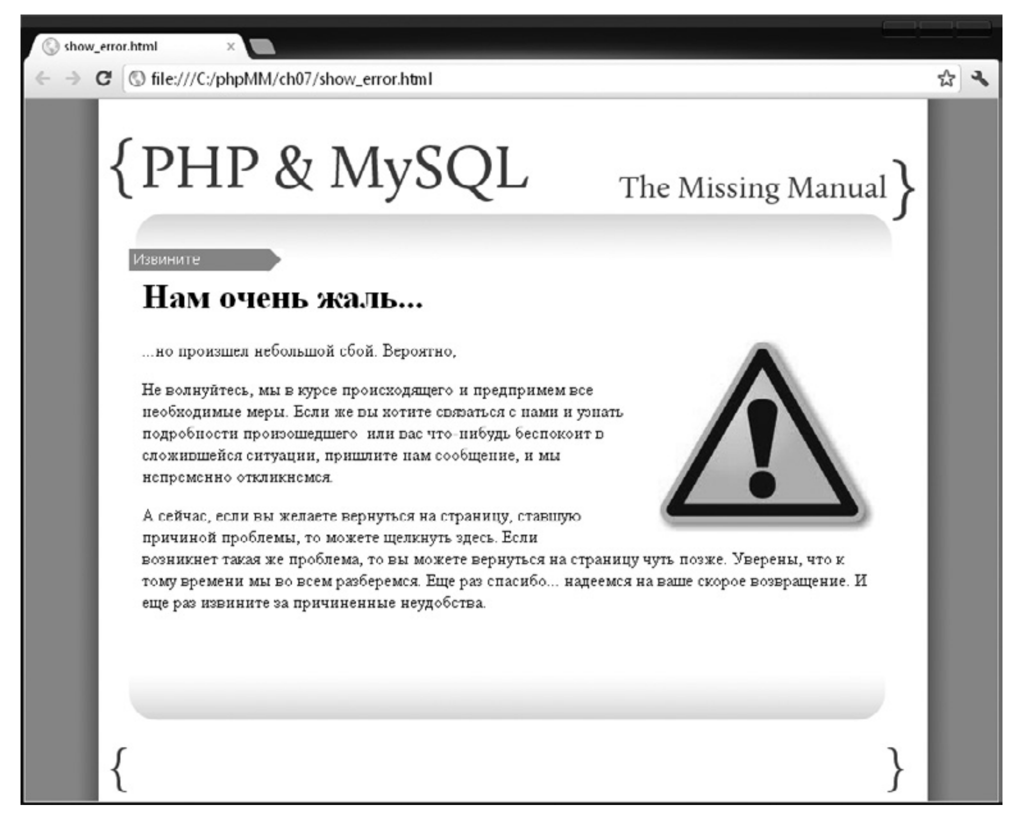


Рис. 7.8. Страница выглядит незавершенной, усугубляя тревожное состояние попавших на нее пользователей

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Почему файл show_error.php находится в каталоге scripts/?

В последней главе вы начнете перемещать свои сценарии из вложенных в каталог scripts/ подкаталогов в основные части своего сайта. При этом вы, возможно, начнете размещать веб-формы вроде `create_user.html` в непосредственной близости от `create_user.php` и `show_user.php`. Причина в том, у ваших HTML- и PHP-страниц появляется больше сходств, чем различий.

Но здесь `show_error.php` все еще располагается в каталоге `scripts/`. Что это дает? Дело в том, что `show_error.php` на самом деле не просто еще одна HTML-страница. Она относится к разряду особенных, используемых по всему вашему приложению. Фактически этим она похожа на файл `database_connection.php`, который также нужно хранить в основном каталоге scripts/. На самом деле они являются обслуживающими (сервисными) программами, а не страницами, которые должны быть поблизости от других HTML-страниц.

> Возникает еще один вытекающий из этого вполне естественный вопрос: не будет ли со временем сплошной путаницы? PHP-файлы находятся рядом с HTML... и что потом? Изображения последуют за JavaScript, а тот в свою очередь за CSS? Так можно по неосторожности свести все к полному абсурду.
>
> В конечном счете здесь нет никакого умысла. Наверное, нужно стремиться к организации ваших файлов по их функциональному назначению. У вас может быть каталог по имени users/, в котором будут находиться все ваши файлы, имеющие отношения к пользователям: `show_user.php`, `create_user.php` и `create_user.html`. У вас могут быть другие подобные каталоги, например groups/ и social/ и т. д.
>
> Когда начинается организация по функциям, структура приобретает осмысленность. Она дает понять, что чем занимается, а не то, что оно собой представляет (CSS, PHP или что-нибудь еще). Вы можете продолжить разбиение, отделяя код, предназначенный для отображения, от кода, который взаимодействует с вашей базой данных. До этого пока еще далеко, но сейчас уже нужно думать о превалировании функционального предназначения над форматом. Важнее сгруппировать файлы, имеющие отношение к пользователям, чем просто PHP-сценарии.
>
> Так что пока храните ваши сервисные сценарии в каталоге scripts/. И, конечно же, если захочется, можно задуматься над переименованием каталога scripts/ в utilities/. Продолжайте совершенствовать структуру, когда у вас будет 50 или 100 файлов, это вам здорово пригодится.

Вы снова вводите пользователя в замешательство, а это плохо. Но есть простое решение: проработать ситуацию, когда параметр запроса отсутствует:

```
<?php
  $error_message = preg_replace_all("/\\\\/", '',
                                   $_REQUEST['error_message']);
  if (!isset($error_message)) {
    $error_message = "вы здесь оказались из-за сбоя в работе программы.";
  }
?>

<html>
  <!-- Существующий HTML и PHP -->
</html>
```

Ранее функция `isset` нам не встречалась, но в ней заложен глубокий смысл: если переменная `$error_message` объявлена или имеет значение, то все хорошо, а если нет (именно этот смысл придает выражению оператор `!`), то переменной `$error_message` присваивается текст сообщения обобщенного характера. Функция `isset` возвращает `true`, если переменной присвоено какое-нибудь значение и оно не равно `null`. В данном случае это нам подходит: даже если вы присваиваете переменной `$error_message` значение, имеющееся в `$_REQUEST['error_message']`. Этим значением может быть `null`, поэтому функция `isset` вполне справляется с возложенной на нее задачей.

Вызовите свою страницу ошибки еще раз, не указывая ничего в URL, и вы опять получите вполне приглядную картину (рис. 7.9).



Рис. 7.9. Очередная страница ошибки

Вы можете, наверное, слегка изменить стиль этой страницы. Курсив больше подходит к конкретному сообщению об ошибке, а здесь он смотрится немного нелепо. Если вы хотите немного поимпровизировать, можете установить класс для span-контейнера, в котором выводится сообщение об ошибке, в зависимости от того, какой характер у этого сообщения, общий или конкретный.

А теперь вас ждут проблемы безопасности и фишинга

Добро пожаловать в решение весьма неприятной большой, проблемы. Когда для передачи информации сценарию используется параметр запроса, передать информацию может кто угодно, включая и злоумышленников. В качестве значения в параметр запроса `error_message` они могут помещать собственные сообщения об ошибке... или вообще могут помещать нечто, вообще не являющееся сообщением об ошибке. А в чем здесь проблема? Читайте дальше.

Фишинг и незаметное перенаправление

Помещение в URL такой актуальной информации, как сообщение об ошибке, открывает двери для такого типа интернет-вандализма, который называется фишингом. *Фишинг* — это технология предоставления пользователю того, что представляется ему надежным URL, но уводящая при этом пользователя на неблагонадежный веб-сайт. Предположим, вы получили по электронной почте сообщение со ссылкой на веб-сайт, имеющей следующий вид:

```
http://yellowtagmedia.com/phpMM/ch07/show_error.php?error_message=
%3Ca%20href=%22http://www.syfy.com/beinghuman%22%3E %F9%E5%EB%EA%ED%E8%F2
%E5%20%E7%E4%E5%F1%FC%2C%20%F7%F2%EE%E1%FB%20%EF%EE%EB%F3%F7%E8%F2%FC%20
%EE%EF%E8%F1%E0%ED%E8%E5%20%EE%F8%E8%E1%EA%E8%3C/a%3E
```

Вы можете просто щелкнуть на этой ссылке. В конце этой ссылки содержится много малопонятной информации, но вы распознаете наиболее важную часть, имя хоста: `yellowtagmedia.com`. В данной книге доменное имя `yellowtagmedia.com` встречалось не один раз. Это домен автора, поэтому вы можете подумать, что этот сайт вполне подходит для посещения. Вы щелкаете на ссылке и видите страницу, как на рис. 7.10.

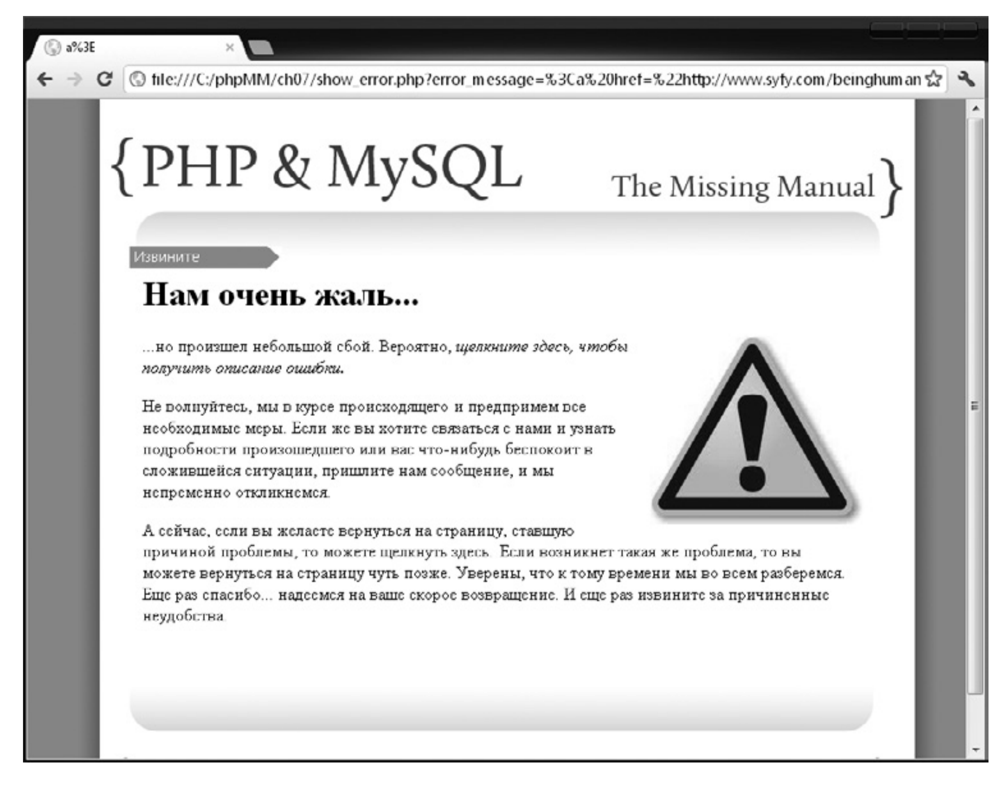


Рис. 7.10. Страница, похожая на страницу ошибки

Эта страница ошибки очень похожа на ту, которую вы уже создавали. И посмотрите, на ней есть ссылка, которой вы также можете довериться, поскольку она появилась на странице, которой вы доверяете. Поэтому вы щелкаете на ссылке и оказываетесь совершенно на другом сайте, на который даже не ожидали попасть (рис. 7.11).



Рис. 7.11. Классический пример фишинга: вы заходите на проверенный сайт (или полагаете, что это так и есть), а оказываетесь на стороннем сайте

Теперь страница канала SyFy с описанием сериала «Быть человеком» (Being Human) будет призывать вас посмотреть этот действительно неплохой фильм. Но представьте себе, что подобная ссылка приведет вас на сайт, запрашивающий номер вашей кредитной карты, или на сайт, заполненный дискредитирующим вас материалом, случайный просмотр которого в рабочее время может привести к вашему увольнению, или даже на простой сайт, запрашивающий «повторное подтверждение» вашего имени пользователя и пароля. Все это таит в себе потенциальные неприятности.

Хитрый и непорядочный программист может запросто воспользоваться теми же CSS, которые применяются на сайте yellowtagmedia.com, чтобы придать своему сайту такой же внешний вид, как и у исходной страницы ошибок, и большинство пользователей не заметит между ними разницы.

Опасность, исходящая от параметров запроса

Любой опытный пользователь может набрать параметр запроса, и это может вызвать проблемы. Вернемся к URL, с которого все началось:

```
http://yellowtagmedia.com/phpMM/ch07/show_error.php?error_message=
%3Ca%20href=%22http://www.syfy.com/beinghuman%22%3E %F9%E5%EB%EA%ED%E8%F2
%E5%20%E7%E4%E5%F1%FC%2C%20%F7%F2%EE%E1%FB%20%EF%EE%EB%F3%F7%E8%F2%FC%20
%EE%EF%E8%F1%E0%ED%E8%E5%20%EE%F8%E8%E1%EA%E8%3C/a%3E
```

Все неприятности исходят от параметра `error_message`. В качестве значения допускается... все, что угодно. И если вернуть обычный вид всем переведенным в коды символам, этот URL приобретет следующий вид:

```
http://yellowtagmedia.com/phpMM/ch07/show_error.php?error_message=<a
href="http://www.syfy.com/beinghuman">щелкните здесь, чтобы получить описание
ошибки </a>
```

Таким образом, ссылка на неблагонадежный сайт прямиком попадает на надежную страницу. Это серьезная проблема, способная создать большие неприятности вашим пользователям.

К сожалению, для ее устранения приходится приложить немалую долю PHP-мастерства, которым вы еще не обладаете. Но оно еще придет... глав через шесть. А сейчас используйте прежний подход к обработке ошибок, но учтите, что для серьезной работы он не годится. Вам нужно будет применить такую технологию, как сессии, подробности которых рассматриваются в главе 13, чтобы не стать когда-нибудь соучастником такого жульничества, как фишинг.

ПРИМЕЧАНИЕ

Какой бы призрачной ни казалась угроза фишинга, но она есть. Чтобы выявить потенциальную проблему, нужен грамотный технический эксперт. Но такова цена создания программ в больших, опасных Всемирных сетях: вы всегда должны быть в курсе того, что какой-нибудь пакостливый тинейджер может сделать с вашим сайтом в случае недосмотра. Но, к счастью, вы изучаете все, что нужно для борьбы и обороны от подобных атак. Нужно потерпеть до главы 13, где будет рассматриваться использование сессий с целью внесения небольших изменений, которые полностью перекроют путь для любых попыток фишинга.

Добавление отладки к приложению

Вы позаботились о своих пользователях на случай возникновения ошибки, но не пора ли позаботиться о самом себе? Вы ведь должны быть в курсе происходящего не только в вашем коде, но и во внешнем интерфейсе. Но созданные вами страницы ошибок теперь закрывают от вас то, что в действительности происходит на уровне сценария. Вместо наблюдения технически выверенных сообщений об ошибках, столь неприглядных и нечитаемых с точки зрения ваших пользователей, вы получаете привлекательные, успокаивающие сообщения. Но для вас-то они абсолютно бесполезны!

И при каждой неблагоприятно сложившейся ситуации вы роетесь во всем своем коде, без четких ориентиров относительно случившейся неприятности. С этим никак нельзя мириться. Лучше определить способ отображения настоящих сообщений о проявившихся ошибках, но сделать это так, чтобы их могли видеть только вы.

Включение отчета об ошибках, выдаваемого интерпретатором PHP

В первую очередь вам нужен отчет об ошибках в момент их проявления, особенно если программа ведет себя странным образом и о ее поведении ничего не сообщается. Рассмотрим, к примеру, следующий фрагмент кода:

```
echo "Привет, {$first_name}\n\n";
$query = "SELECT * FROM users WHERE first_name = {$first_name}";
```

Здесь результат радикально изменяется в зависимости от того, есть у переменной `$first_name` значение или нет. Могут возникать ошибки обращения к базе данных, возвращаться странные результаты запросов или случаться более существенные неприятности. Теперь, конечно же, для решения проблемы вы можете добавить несколько вызовов функции `isset`, но зачастую, пока что-нибудь пойдет не так, о таком методе предотвращения ошибок обычно забывают. Вам нужно получить не привлекательное сообщение об ошибке, а отчет при возникновения нештатной ситуации. Тогда вы сможете внести необходимые поправки и избежать повторения подобных ошибок.

И здесь свою помощь предлагает интерпретатор PHP: вы можете включить в нем самом *отчет об ошибках*. Обычно это делается с помощью файлов низкоуровневой конфигурации, используемых PHP, но это слишком сложное решение, чем того требует наша ситуация.

ПРИМЕЧАНИЕ

Многие интернет-провайдеры и компании, предоставляющие веб-хостинг, не позволят вам приблизиться к конфигурационным файлам веб-серверов и того PHP, который на них установлен. Они не хотят подвергать опасности себя и поддерживаемое ими оборудование.

Чтобы увидеть все это в действии, создайте небольшой сценарий по имени `display_error.php` и наберите следующий код:

```
<?php

echo "Привет, {$first_name}\n\n";
$query = "SELECT * FROM users WHERE first_name = {$first_name}";
echo "{$query}\n\n";

?>
```

Понятно, что в нем есть какая-то неопределенность с проблемой, которая связана с переменной $first_name. И хотя этот сценарий не собирается выполнять запрос, который будет неполным, явно это та самая программа, в которой вы хотите знать, что происходит нечто нехорошее.

Запустите эту программу, и вы получите следующий результат:

```
$ php display_error.php
Привет,

SELECT * FROM users WHERE first_name =
```

Весьма корявый результат, не так ли? PHP спокойно выполняет программу, игнорируя проблему. Стало быть, вы не будете перенаправлены ни на какую страницу ошибки, по крайней мере если только через несколько программных строк не воспользуетесь этим же запросом в отношении своей базы данных. Но к тому времени вы уже на несколько строк кода (а может быть, даже на несколько сотен строк!) отдалитесь от места реальной проблемы, заключающейся в утрате значения для переменной $first_name.

И здесь настает черед PHP-функции error_reporting. Добавьте в сценарий display_error.php следующую строку кода:

```
<?php

error_reporting(E_ALL);

echo "Привет, {$first_name}\n\n";
$query = "SELECT * FROM users WHERE first_name = {$first_name}";
echo "{$query}\n\n";

?>
```

Константа E_ALL задает уровень отчета. Благодаря E_ALL выдаются отчеты о любой возможной ошибке. Можно также воспользоваться константами E_ERROR, E_WARNING, E_PARSE и E_NOTICE, каждая из которых задает выдачу отчетов о разных ошибках (молча позволяя совершаться другим ошибкам). Полный перечень различных уровней можно получить на сайте **www.php.net/manual/en/function.errorreporting.php**. Но в самом простом случае константа E_ALL задает выдачу исчерпывающих сведений о нештатных ситуациях.

Запустите сценарий еще раз, и вы получите совершенно другой результат:

```
$ php display_error.php
PHP Notice:  Undefined variable: first_name in yellowtagmedia_com/phpMM/ch07/
display_error.php on line 5

Notice: Undefined variable: first_name in yellowtagmedia_com/phpMM/ch07/dis-
play_error.php on line 5
Привет,

PHP Notice:  Undefined variable: first_name in yellowtagmedia_com/phpMM/ch07/
```

```
display_error.php on line 6

Notice: Undefined variable: first_name in yellowtagmedia_com/phpMM/ch07/dis-
play_error.php on line 6
SELECT * FROM users WHERE first_name =
```

Неожиданно интерпретатор PHP стал сверхосведомленным о потенциальных проблемах, позволяя вам узнать об их существовании. Это поможет довести ваше приложение до работоспособного состояния, и вы сможете узнать о потенциальной возможности появления ошибок. Что ж, неплохо.

ВНИМАНИЕ

Переоценить роль таких отчетов для написания хорошего кода, конечно, трудно, но они все же являются раздражающим фактором. Вы обрекаете себя на постоянные напоминания со стороны интерпретатора PHP о ваших потенциальных просчетах. Тем не менее это весьма умеренная плата за знание о потенциальных проблемах, имеющихся в вашем сценарии.

Глобальное включение выдачи отчетов об ошибках

Осталась еще одна проблема: не забыть включить выдачу отчетов об ошибках. Но делать это для каждого сценария не хотелось бы. Тем более что вы собираетесь обзавестись не одной такой основной процедурой, как выдача отчетов об ошибках, которую нужно применить ко всем сценариям.

Возможно, решение этого вопроса для вас уже стало вполне очевидным: нужен еще один сценарий такого же типа, как database_connection.php, который будет отвечать за подобное общее поведение. Тогда все другие ваши сценарии смогут включать это поведение с помощью одного-единственного вызова, который возьмет на себя все остальное. Но у вас уже есть подобный файл: app_config.php, который используется сценарием database_connection.php для таких общих констант, как имя вашей базы данных и пароль. Именно это вам сейчас и нужно.

ПРИМЕЧАНИЕ

Да, это действительно означает, что вы по-прежнему должны включать этот один общий сценарий app_config.php во все остальные ваши сценарии. И по этому поводу следует сделать себе какое-нибудь напоминание. Подробнее эта тема рассмотрена в следующей врезке «Курсы повышения квалификации».

Продолжим работу и откроем файл app_config.php, который находится в корневом каталоге scripts/. Там же должны располагаться файлы show_error.php и database_connection.php. Добавьте в него инструкцию error_reporting, чтобы включить выдачу отчетов об ошибках для всех ваших сценариев:

```
<?php

// Константы подключения к базе данных

// Включение выдачи отчетов об ошибках
error_reporting(E_ALL);

?>
```

Теперь нужно лишь включить вызов этого сценария во все свои сценарии:

```
<?php

require '../scripts/app_config.php';

echo "Привет, {$first_name}\n\n";
$query = "SELECT * FROM users WHERE first_name = {$first_name}";
echo "{$query}\n\n";

?>
```

ПРИМЕЧАНИЕ

Если вы выполняете все, о чем здесь говорится, то должны убрать функцию error_reporting из сценария display_error.php, поскольку теперь она запускается в сценарии app_config.php.

Теперь с таким добавлением вы станете получать отчеты об ошибках при работе всех своих сценариев. Весьма полезное обновление ценой всего лишь одной строки кода, добавленной к каждому вашему сценарию.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Необходимость запоминать те или иные вещи, является (в какой-то мере) частью программирования

Итак, похоже, что после всех разговоров о том, как бы не забыть включить выдачу отчетов об ошибках, мы пришли к решению, заключающемуся в том, чтобы... не забыть включить в сценарии вызов app_config.php. Так что же здесь, собственно, упростилось?

К сожалению, как программисту вам всегда приходится помнить о необходимости совершения каких-нибудь конкретных действий. Это может быть избавление от некой конкретной переменной, отнимающей массу системных ресурсов, или отключение от базы данных, или выход из системы, или включение какого-нибудь файла в каждый создаваемый вами сценарий... Главное при этом — сократить объем того, о чем нужно помнить. Это как раз и касается использования `app_config.php`. При включении только одного этого файла (для чего нужно лишь одно напоминание) вы сможете включить все то общее, что нужно для ваших сценариев. Поэтому если впоследствии вам понадобится дополнить содержимое файла `app_config.php`, то это дополнение попадет сразу во все ваши сценарии. (И совсем скоро нам понадобится именно такое дополнение к `app_config.php`.) Поэтому при любой такой возможности делайте в сценарии всего одно включение, а не два, три или десять.

Выключение выдачи отчетов об ошибках при переходе в эксплуатационный режим

Теперь выдача отчетов об ошибках у вас включена, и объем получаемой информации существенно возрос. Но есть еще одна проблема: иногда отчет касается не ошибки, а потенциальной возможности ее возникновения. В качестве примера обеспечьте включение `app_config.php` в свой сценарий `show_error.php`:

```
<?php
  require 'app_config.php';

  $error_message = preg_replace_all("/\\\\/", '',
                                   $_REQUEST['error_message']);
  if (!isset($error_message)) {
    $error_message = "вы здесь оказались из-за сбоя в работе программы.";
  }
?>

<html>
  <!-- HTML и PHP -->
</html>
```

Теперь войдите на страницу show_error.php через свой браузер, не помещая ничего в качестве сообщения об ошибке. Для сценария show_error.php это не станет проблемой, поскольку код предусматривает подобную ситуацию. Но посмотрите на рис. 7.12, подобный вывод из сценария может стать для вас сюрпризом.

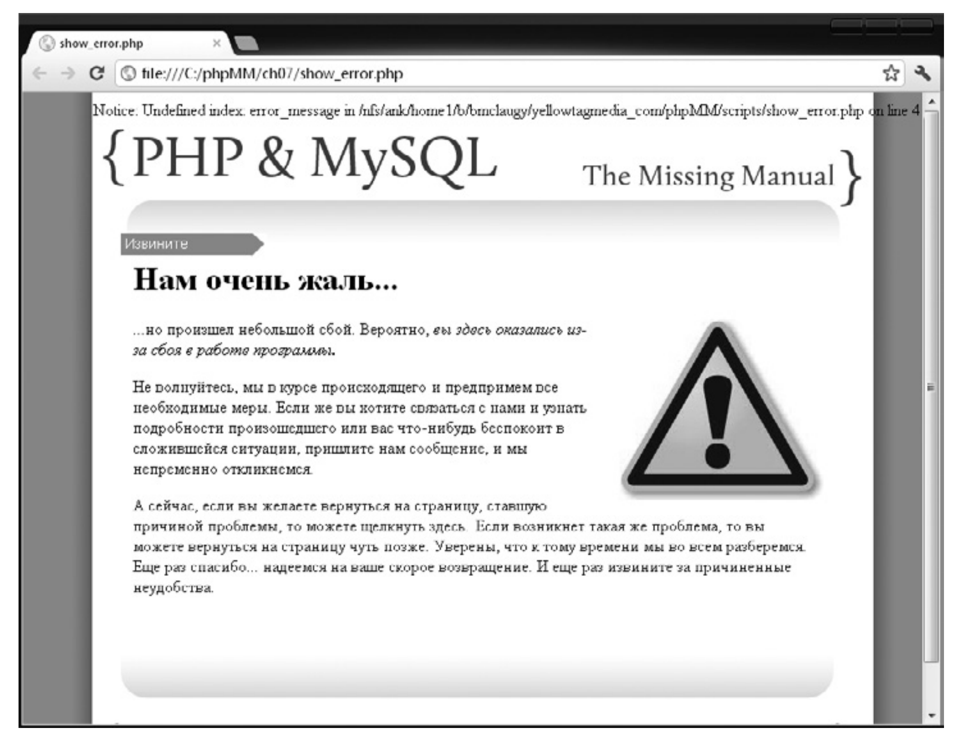


Рис. 7.12. Внезапно появилась наихудшая из возможных страниц ошибок: у нее есть своя собственная ошибка!

При всем при том, что вы контролируете ситуацию, технически существует потенциальная проблема, заключающаяся в том, что вы присваиваете переменной $error_message значение ($_REQUEST['error_message']), которое может быть пустым (null).

С этой проблемой можно справиться двумя вполне подходящими способами. Вы можете реорганизовать этот код, избегая прямого доступа к потенциально пустому (null) значению:

```
<?php
  require 'app_config.php';

  if (isset($_REQUEST['error_message'])) {
    $error_message = preg_replace_all("/\\\\/", '',
                                      $_REQUEST['error_message']);
  } else {
    $error_message =
      "вы здесь оказались из-за сбоя в работе программы.";
  }
?>

<html>
  <!-- HTML и PHP -->
</html>
```

Теперь у интерпретатора PHP отсутствует проблема при использовании значения null в функции isset. Фактически isset для того и предназначена, чтобы избавить вас от применения неожиданного значения null. Поэтому в данном смысле выдача отчета об ошибке помогла вам усовершенствовать эту страницу. Перезагрузите страницу, и будет выведено сообщение об ошибке, принявшее свой прежний вид (см. рис. 7.9).

Но за всем этим кроется более серьезная проблема: даже притом, что это незначительное изменение улучшило ваш код, могут быть случаи, когда приходится допускать контакт пользователей с вашей системой еще до того, как она достигнет стопроцентного совершенства. Кто-то из мудрецов сказал: «Лучшее — враг хорошего». Можно расширить это высказывание: «Совершенное веб-приложение — враг хорошего веб-приложения». Если дожидаться, пока ваш код станет идеальным, вы, наверное, никогда не запустите приложение в эксплуатацию.

И что же делать? Представьте, что вы можете установить режим работы своего приложения. То есть вы можете запустить его в режиме отладки, и будут выводиться отчеты об ошибках, или в режиме эксплуатации, и режим выдачи отчетов об ошибках не включится. Затем вы сможете просто запустить его в отладочном режиме, пока не наступит время его реальной работы, а затем переключитесь в эксплуатационный режим.

ПРИМЕЧАНИЕ

Вы можете даже пойти дальше: скопировать код на сервер, переключить его на работу в эксплуатационном режиме, а затем запустить другую копию в отладочном режиме для работы по ее совершенствованию.

Установка отладочного режима не представляет особого труда. А при наличии сценария app_config.php вы уже имеете отличное общее место для настройки подобных режимов:

```
<?php

// Установка режима отладки
define("DEBUG_MODE", true);

// Константы подключения к базе данных

// Выдача отчетов об ошибках
if (DEBUG_MODE) {
  error_reporting(E_ALL);
} else {
  // Выключение выдачи отчетов об ошибках
  error_reporting(0);
}

?>
```

Вот и все. Теперь у вас есть возможность внести всего одно изменение в DEBUG_MODE и получить (или не получить) выдачу отчетов об ошибках во всем приложении.

Переход от require к require_once

Если вы при создании своего программного кода строго придерживались изложенных здесь инструкций, то в верхней части сценария database_connection.php можете увидеть такую строку:

```
require 'app_config.php';
```

Следовательно, любой сценарий, в котором выполняется следующая инструкция:

```
require '../../scripts/database_connection.php';
```

автоматически требует также сценарий app_config.php. Стало быть, если вам нужны настройки из файла app_config.php в сценарии, который уже затребовал сценарий database_connection.php, то с технической точки зрения сценарий app_config.php требовать в явном виде не нужно.

Но (и это довольно весомое «но») теперь зависимость в вашем коде носит скрытый характер. Даже притом, что вы не требуете app_config.php в явном виде, вы пишете код, предполагающий, что app_config.php уже был загружен. А теперь представьте, что вы изменили сценарий и больше не используете в нем базу данных. В таком случае вполне естественно следующим шагом удалить строку с требованием загрузки сценария database_connection.php. Ведь если ваш сценарий больше не пользуется базой данных, запрос сценария database_connection.php уже не будет иметь смысла. Но удаляя этот запрос, вы также теряете запрос и сценария app_config.php. При этом возникает проблема, которая ничем себя не проявляет, пока не обнаружится отсутствие нужных констант и определений сообщений об ошибках.

По одной только этой причине неплохо было бы выразить свои требования в явном виде. Но теперь возникают вполне справедливые опасения: вы запросите сценарий

app_config.php, а затем запросите также сценарий database_connection.php, который в свою очередь запрашивает app_config.php. Получается, что в сценариях, работающих с базой данных, вы запрашиваете app_config.php дважды. И тут возникает проблема, потому что в результате двойной загрузки константы определяются дважды, что приводит к выдаче интерпретатором PHP ошибки:

```
// Константы подключения к базе данных
define("DATABASE_HOST", "db.host.com");
define("DATABASE_USERNAME", "username");
define("DATABASE_PASSWORD", "super.secret.password");
define("DATABASE_NAME", "db-name");
```

А вот сообщение об ошибке, которое вы увидите, если дважды воспользуетесь в этом файле инструкцией require в отношении одного и того же сценария:

```
Notice: Constant DATABASE_HOST already defined in yellowtagmedia_com/phpMM/
scripts/app_config.php on line 4
Notice: Constant DATABASE_USERNAME already defined in yellowtagmedia_com/
phpMM/scripts/app_config.php on line 5
Notice: Constant DATABASE_PASSWORD already defined in yellowtagmedia_com/
phpMM/scripts/app_config.php on line 6
```

Чтобы справиться с данной дилеммой, можно во всех сервисных сценариях вместо инструкции require воспользоваться инструкцией require_once. То есть в вашем основном сценарии, там, где находится основной код, применяется обычная инструкция require:

```
// основной сценарий, в который помещается код
require '../scripts/app_config.php';
```

А в любом сервисном сценарии, которому также нужен сценарий app_config.php, используется инструкция require_once:

```
// database_connection.php и любой другой сервисный сценарий
require_once '../scripts/app_config.php';
```

Инструкция require_once проверяет, не включался ли уже указанный сценарий (путем использования инструкции include или require), и включает этот сценарий только в том случае, если он еще не был загружен. Это гарантирует однократную загрузку сценария app_config.php.

Но здесь возникает еще одна проблема: иногда один сценарий, вроде create_user.php, вызывает другой сценарий, например show_user.php. В таком случае у вас оказываются в наличии два сценария, оба из которых, возможно, используют инструкцию require, в результате чего вы будете получать ошибки, связанные с переопределением констант. Нужно ли тогда передумать и реорганизовать код сценария app_config.php? Необходимо ли выделить эти константы в другой файл или переместить их в сценарий database_connection.php?

Вообще-то, вы можете обойти все эти проблемы путем использования инструкции require_once во всех своих сценариях. Таким образом будет гарантировано, что сце-

нарий app_config.php никогда не будет загружен более одного раза. Но есть еще один побочный эффект: теперь не нужно будет определять, какую версию инструкции require надо применять. Нужно взять за правило повсеместно использовать инструкцию require_once, пока у вас не возникнут специфические потребности в многократной загрузке какого-нибудь файла. Поскольку что-либо требуется дважды довольно редко, применение данного правила имеет вполне определенный смысл.

ПРИМЕЧАНИЕ

Да, в этой книге от вас не требовалось с самого начала использовать функцию require_once, поскольку тогда вы бы не поняли, почему она применяется вместо функции require. Пройдя весь этот процесс выявления ошибок, теперь вы можете объяснить своим коллегам, почему им также нужно в своих PHP-сценариях задействовать инструкцию require_once.

Сейчас вы меня видите, а сейчас нет

К сожалению, проделав большой объем работы, вы по-прежнему не решили одной ключевой проблемы: нужен такой способ вывода, котрый будет содержать более подробную информацию об ошибке для вас и ваших друзей-программистов и не пугать при этом ваших пользователей. Но некоторая основа для этого уже заложена. В созданном вами файле app_config.php имеется константа DEBUG_MODE, которая может стать ключевым «ингредиентом».

Теперь вам нужен способ вывода дополнительной информации об ошибках при нахождении в отладочном режиме. Здесь, как и в вопросе выдачи отчетов об ошибках (посредством собственного обработчика ошибок, имеющегося в PHP), вы можете просто выключить эту функцию в режиме эксплуатации. Впоследствии вы всегда сможете включать ее на короткий период в случае возникновения проблемы, а затем снова выключать после получения отчета об ошибке, позволяющего обнаружить и устранить любую случившуюся ошибку.

Итак, определим новую функцию под названием debug_print, которая выводит информацию только в отладочном режиме:

```
function debug_print($message) {
  if (DEBUG_MODE) {
    echo $message;
  }
}
```

Если она будет определена в файле app_config.php, доступ к ней можно будет получить из любого вашего кода. Она всего лишь избирательно выводит сообщение: если отладка включена, она его выводит, а если нет, значение переменной $message на экран не попадет.

Вы только что создали свою первую пользовательскую функцию! Превосходно. Хотя вам еще предстоит многое узнать о пользовательских функциях, заметьте, как просто создается собственная манера поведения для использования во всем остальном вашем приложении.

Теперь вы можете добавить в страницу `show_error.php` некоторую дополнительную информацию:

```
<?php
  require 'app_config.php';

  if (isset($_REQUEST['error_message'])) {
    $error_message = preg_replace_all("/\\\\/", '',
                                      $_REQUEST['error_message']);
  } else {
    $error_message = "вы здесь оказались из-за сбоя в работе программы.";
  }

  if (isset($_REQUEST['system_error_message'])) {
    $system_error_message = preg_replace("/\\\\/", '',
                            $_REQUEST['system_error_message']);  } else {
    $system_error_message = "Сообщения о системных ошибках отсутствуют.";
  }
?>
```

ПРИМЕЧАНИЕ

Есть и другие способы получения такого же результата. Но если включена функция error_reporting, то при использовании данного способа характер каких-либо ошибок уточняться не будет.

Затем в недрах своего HTML выборочно наберите следующую дополнительную информацию:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Извините</div>

  <div id="content">
    <h1>Нам очень жаль...</h1>
    <!-- Существующие для пользователя обработка ошибки и вывод
         информации -->
    <?php
      debug_print("<hr />");
      debug_print("<p>Было получено следующее сообщение об ошибке системного
        уровня:
<b>{$system_error_message}</b></p>");
    ?>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

И в завершение вы можете собрать все вместе. У вас уже есть страница ошибки, средства вывода информации только при включенном отладочном режиме и сценарий app_config.php. И теперь вы можете связать все это вместе.

Переадресация на ошибку

Теперь у вас есть довольно сложный механизм работы с сообщениями об ошибках в случае их возникновения, и у вас даже есть способ получения отчета об ошибках от PHP (посредством функции error_reporting), а также средства вывода ошибок для облегчения процесса программирования (с помощью функции debug_print). Но вы еще всем этим не пользовались. Настало время исправить данную ситуацию.

Посмотрите на одну из ваших простейших комбинаций страница-сценарий из главы 4: connect.html и connect.php.

ПРИМЕЧАНИЕ

Скопируйте эту комбинацию в новый каталог, чтобы в них можно было вносить изменения. Затем вы должны изменить connect.html для отправки данных сценарию connect.php без использования каталога scripts/ и обеспечить расположение connect.php в непосредственной близости от connect.html. Необходимо также обеспечить наличие инструкций require_once app_config.php и отображение в пути к app_config.php нового местоположения файла connect.php. Звучит так, будто предстоит огромный труд, но хорошенько просмотрите свой код PHP, и поймете, что все элементарно.

Обновление вашего сценария для использования show_error.php

Вот первое место, где требуются изменения:

```
<?php

  require '../scripts/app_config.php';

  mysql_connect(DATABASE_HOST, DATABASE_USERNAME, DATABASE_PASSWORD)
    or die("<p>Ошибка подключения к базе данных: " .
           mysql_error() . "</p>");

  // и т. д...
?>
```

Если прямо сейчас функция mysql_connect даст сбой, то будет прекращена работа всего сценария. А это нас не устраивает. Теперь одним из способов возможного исправления данной ситуации будет использование примерно следующего кода:

```
if (!mysql_connect(DATABASE_HOST,
                   DATABASE_USERNAME, DATABASE_PASSWORD)) {
  $user_error_message = "возникла проблема, связанная с " .
                        "подключением к базе данных, " .
                        "содержащей нужную информацию.";
  $system_error_message = mysql_error();
  header("Location: ../scripts/show_error.php?" .
```

```
        "error_message={$user_error_message}&" .
        "system_error_message={$system_error_message}");
  exit();
}
```

ПРИМЕЧАНИЕ

Это один из фрагментов кода, использующий длинные строки, которые не помещаются по ширине книжной стрницы. Вам, конечно же, не нужно разбивать такие строки на несколько строк, но при желании вы можете это сделать. В загружаемом коде для определения значения переменной $user_error_message, а также для предоставления функции header URL-адреса используется одна строка.

То, что применяется на вашей новой странице ошибок в сочетании в PHP-перенаправлением, предоставляет привлекательное сообщение об ошибке, а также сообщение на системном уровне и будет работать вполне исправно. Чтобы протестировать работу, наберите неприемлемый хост базы данных в следующем формате:

```
if (!mysql_connect(DATABASE_HOST, DATABASE_USERNAME, "foo")) {
  // обработка ошибки
}
```

Теперь выйдите на `connect.html` в своем браузере, отправьте форму на `connect.php`. В результате появится страница ошибки (рис. 7.13).

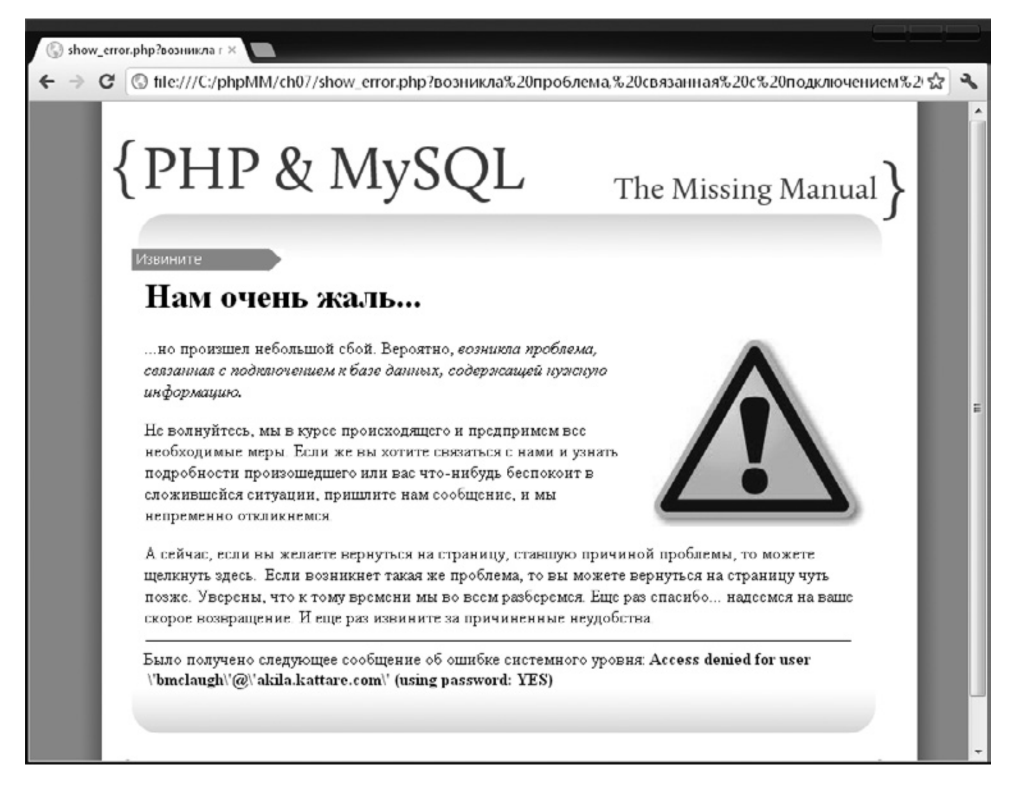


Рис. 7.13. Теперь вы видите то же самое, что видит ваш пользователь, плюс получаете отчет об ошибке на программном уровне

ПРИМЕЧАНИЕ

Перед тем как это опробовать, убедитесь, что константа DEBUG_MODE в файле app_config.php имеет значение true, чтобы вы могли увидеть оба вида сообщений об ошибках, предназначенные как для пользователей, так и для разработчиков.

Все замечательно! Вы добились того, что с точки зрения сообщения об ошибках ваши пользователи полностью удовлетворены. Теперь установите для константы DEBUG_MODE в файле app_config.php значение false:

```
// Установка режима отладки
define("DEBUG_MODE", false);
```

Перейдите еще раз в connect.html и connect.php, на этот раз вы должны увидеть только сообщения об ошибках, предназначенные для пользователей (рис. 7.14).



Рис. 7.14. Сообщение об ошибке, которое видит пользователь

Вот какая разная картина получается в результате совершенствования приемов обработки ошибок. Вы помните, как ошибка базы данных приводила к появлению пустой страницы с каким-то зашифрованным сообщением об ошибке? А теперь, после весьма существенного обновления, сообщения об ошибках оформляются в привлекательном стиле, и это является весьма важным обстоятельством для рядового пользователя.

Простота и абстракция

Итак, задача решена? Почти. Вывод сообщения об ошибке получился неплохо, но взгляните еще раз на код в основном сценарии connect.php:

```
if (!mysql_connect(DATABASE_HOST,
                   DATABASE_USERNAME, DATABASE_PASSWORD)) {
  $user_error_message = "возникла проблема, связанная с " .
                        "подключением к базе данных, " .
                        "содержащей нужную информацию.";
  $system_error_message = mysql_error();
  header("Location: ../scripts/show_error.php?" .
         "error_message={$user_error_message}&" .
         "system_error_message={$system_error_message}");
  exit();
}
```

Здесь много кода, занятого обработкой проблемной ситуации. Причем его даже больше, чем кода, занятого штатной ситуацией. Нельзя сказать, что это при любых обстоятельствах плохо, но в данном случае в этом нет особой необходимости. Помните, как этот код выглядел изначально?

```
mysql_connect(DATABASE_HOST, DATABASE_USERNAME, DATABASE_PASSWORD)
  or die("<p>Error connecting to database: " . mysql_error() . "</p>");
```

Все здесь предельно оптимально — строка, выполняющая требуемое действие, а затем строка на случай возникновения проблем. А теперь умножьте это на количество разных мест, в которых может произойти сбой, получится довольно объемный код обработки ошибок.

Можно ли получить более элегантный вид для кода обработки ошибок? Наверное, стоит попробовать. Приглядитесь к коду еще раз и обратите внимание на то, что независимо от характера ошибки части этого кода будут всегда одинаковыми:

```
if (!mysql_connect(DATABASE_HOST,
                   DATABASE_USERNAME, DATABASE_PASSWORD)) {
  $user_error_message = "возникла проблема, связанная с " .
                        "подключением к базе данных, " .
                        "содержащей нужную информацию.";
  $system_error_message = mysql_error();
  header("Location: ../scripts/show_error.php?" .
         "error_message={$user_error_message}&" .
         "system_error_message={$system_error_message}");
  exit();
}
```

Итак, единственное, что здесь изменяется, — сами сообщения об ошибках. Все остальное — имена переменных, вызов функции header и создание URL — остается неизменным. А что если для работы со всем этим создать еще одну функцию, похожую на debug_print?

Добавьте эту функцию к сценарию app_config.php, продолжив тем самым расширение своего сервисного сценария:

```
<?php
  // Установка режима отладки

  // Константы подключения к базе данных

  // Выдача отчетов об ошибках

function debug_print($message) {
  if (DEBUG_MODE) {
    echo $message;
  }
}

function ($user_error_message, $system_error_message) {
  header("Location: show_error.php?" .
         "error_message={$user_error_message}&" .
         "system_error_message={$system_error_message}");
  exit();
}
?>
```

На самом деле этот сценарий является вариантом того, что вы сделали с debug_print. Вы взяли одинаковый, многократно повторяющийся код и поместили его в очень удобную пользовательскую функцию, на которую можно сослаться в любой момент. Единственным различием является добавление функции exit.

Тем самым гарантируется, что независимо от конкретной структуры вызывающего сценария, поскольку функция header перенаправляет браузер на вашу страницу ошибки, больше ничего не происходит. Выводится страница ошибки, и PHP останавливает все, что могло планироваться к выполнению.

Вы можете немного упростить connect.php:

```
if (!mysql_connect(DATABASE_HOST, DATABASE_USERNAME, "foo")) {
  handle_error("возникла проблема, связанная с подключением к базе данных, " .
              "содержащей нужную информацию.",
              mysql_error());
}
```

Выглядит намного лучше, особенно если учесть, что это всего лишь одна строка в терминале или в редакторе. Но можно продолжить упрощение:

```
mysql_connect(DATABASE_HOST, DATABASE_USERNAME, "foo")
  or handle_error("возникла проблема, связанная с подключением к базе данных, " .
                 "содержащей нужную информацию.",
                 mysql_error());
```

Здесь вы избавились от if и вернулись к простой и привычной элегантности or die, но уже с намного более подходящей функцией — своей собственной handle_error.

Переадресация не видит пути к файлу

Но есть одна проблема, показанная на рис. 7.15.



Рис. 7.15. Иногда PHP сообщает об ошибке совершенно неинформативно

При попытке самостоятельного запуска connect.php вы можете увидеть только такую картину. Хотя она отображает некую нештатную ситуацию, это, конечно же, не та страница show_error.php, над которой вы так упорно трудились. Но что же это такое?

На самом деле это хорошо знакомая ошибка, относящаяся к PHP. Большинство веб-серверов настроены на то, чтобы рассматривать любой URL-запрос, оканчивающийся на .php, как PHP-запрос. Это хорошо с той точки зрения, что вам не нужно скапливать все свои PHP-сценарии в одном каталоге. Но это также плохо, поскольку веб-сервер не видит, что URL, оканчивающийся на .php, соответствует реально существующему файлу. Он просто передает URL PHP-программе. Но этот URL не является указателем на реальный файл. PHP говорит: «Мне нечего запускать». Или, точнее: No input file specified (Входной файл не указан).

Остается вопрос: почему было получено это сообщение? Причина кроется в фрагменте кода сценария app_config.php:

```
function handle_error($user_error_message, $system_error_message) {
  header("Location: show_error.php?" .
         "error_message={$user_error_message}&" .
         "system_error_message={$system_error_message}");
}
```

В этом фрагменте путь к файлу show_error.php задан относительно app_config.php. Поскольку app_config.php находится в том же самом каталоге, что и show_error.php, перед именем файла ничего нет.

Но этот код выполняется из вашего сценария connect.php, который (по крайней мере в примерах из этой книги) находится в каталоге ch07/. Следовательно, путь из этого места к show_error.php имеет вид ../scripts/show_error.php. И даже притом, что функция handle_error определена в app_config.php, она запускается из контекста сценария connect.php. И что получается в результате? Поиск show_error.php осуществляется не в том месте.

Но если изменить путь в app_config.php для работы с connect.php, а затем у вас будет другой сценарий в другом месте, то вы, скорее всего, снова получите ту же проблему. Можно ли тогда по-прежнему считать handle_error сервисной функцией?

Вам опять нужен способ, чтобы указать общее свойство, — корневой каталог своего сайта, а затем связать путь к show_error.php с корневым каталогом, используя не относительный, а абсолютный путь.

К ВАШЕМУ СВЕДЕНИЮ

Относительные и абсолютные пути

Относительным называется путь, который ссылается на файл относительно текущего файла. Обычно это означает, что путь начинается либо от самого файла, такого как `show_error.php`, либо от предыдущего каталога с использованием индикатора в виде двух точек (..). Относительные пути имеют вид `show_error.php` или `../scripts/show_error.php`. В обоих случаях отправной точкой служит текущий файл, показывающий путь.

Абсолютным называется путь, приведенный относительно не текущего файла, а корневого каталога вашего сайта. Признаком абсолютного пути всегда служит начальный символ /, указывающий, что искомый файл нужно искать с корня, или «базы» вашего веб-сайта. Абсолютный файл имеет вид `/scripts/show_error.php`.

Вы можете определить корневой каталог вашего сайта в `app_config.php` с помощью новой константы:

```
// Корневой каталог сайта
define("SITE_ROOT", "/phpMM/");
```

Теперь эту константу можно использовать в функции `handle_error`. Вот как выглядит окончательная версия `app_config.php` со всеми новыми константами и полностью завершенными функциями `handle_error` и `debug_print`:

```
<?php

// Установка режима отладки
define("DEBUG_MODE", false);

// Корневой каталог сайта
define("SITE_ROOT", "/phpMM/");

// Константы подключения к базе данных
define("DATABASE_HOST", "database.host.com");
define("DATABASE_USERNAME", "username");
define("DATABASE_PASSWORD", "super.secret.password");
define("DATABASE_NAME", "database-name");

// Выдача отчетов об ошибках
if ($debug_mode) {
  error_reporting(E_ALL);
} else {
  // Выключение выдачи отчетов об ошибках
  error_reporting(0);
}

function debug_print($message) {
  if (DEBUG_MODE) {
```

```
    echo $message;
  }
}

function handle_error($user_error_message, $system_error_message) {
  header("Location: " . SITE_ROOT . "scripts/show_error.php?" .
         "error_message={$user_error_message}&" .
         "system_error_message={$system_error_message}");
}

?>
```

ПРИМЕЧАНИЕ

Для вставки значений констант в строку можно воспользоваться фигурными скобками, то есть вы должны объединить SITE_ROOT со строкой вашего URL в вызове функции header с помощью оператора точки (.).

Теперь в конечном итоге вы должны получить возможность видеть итоги выполнения сценария show_error.php при возникновении ошибки в connect.php! Сравните результаты всей этой работы с изображением, показанным на рис. 7.14.

В завершение пройдитесь по всем своим сценариям и замените все инструкции die и другие вызовы обработки ошибок вызовами функции handle_error. Не забудьте также обновить сценарий database_connection.php, чтобы в нем использовалась функция handle_error:

```
<?php
  require 'app_config.php';

  mysql_connect(DATABASE_HOST, DATABASE_USERNAME, DATABASE_PASSWORD)
   or handle_error("возникла проблема, связанная с подключением к базе данных, " .
                   "содержащей нужную информацию.",
                   mysql_error());

  mysql_select_db(DATABASE_NAME)
    or handle_error("возникла проблема с конфигурацией нашей базы данных.",
                    mysql_error());
?>
```

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Неужели обработке ошибок посвящено более двух десятков страниц?

В это трудно поверить, неужели это правда? Вы же не добавили к своему веб-приложению никаких дополнительных функций. Конечно, узнали кое-что новое о константах, определили две пользовательские функции, добавили в свой арсенал класс сервисных функций, научились пользоваться инструкциями require и require_once и даже добавили к своему репертуару внутренний PHP-механизм сообщения об ошибках.

Почему-то зачастую тема обработки ошибок располагается в последней главе. При этом считается, что она вообще может быть проигнорирована. Зачем же в принципе тратить время на то, что пользователи, скорее всего, никогда не увидят? Главным образом затем, что приложение, не обрабатывающее ошибок, просто нельзя считать завершенным.

Начиная заниматься программированием или программируя на новом языке, вы в любом случае не сможете избежать ошибок.

Тестирование и обработка ошибок являются наилучшими способами раннего обнаружения ошибок с последующим предоставлением простейшего способа их исправления. Располагая надежной системой обработки ошибок, вы будете удивлены тем, насколько часто серьезная проблема превращается в несерьезную только из-за быстрого обнаружения ошибки и возможности ее отследить, не перелопачивая весь свой код в безуспешных попытках определить причину сбоя.

8 Обработка изображений и решение более сложных задач

Вы подошли к настоящему водоразделу в своей программистской карьере. До сих пор вы использовали множество PHP-конструкций, от инструкций `if` до некоторых основных функций для обработки констант и даже ошибок, и вам стали знакомы основные приемы взаимодействия с MySQL, необходимые в обычных PHP-сценариях. С тем, что вам уже известно, вы готовы взяться за большинство основных задач программирования, с которыми придется сталкиваться в типичных веб-приложениях, которые рассчитаны на использование отдельных страниц.

Иными словами, если у вас есть форма для сбора информации, вы можете ее обработать. Вы можете брать информацию из таблицы, а также помещать ее в таблицу. С этим нет никаких проблем. Вы можете реагировать на ошибки, перенаправлять пользователей и даже отличать хорошее пользовательское восприятие от плохого.

Но вы понимаете, что веб-приложения представляют собой нечто большее, чем сумма их одностраничных взаимодействий. Десять разных страниц, взаимодействующих с десятью различными таблицами, намного проще сложного десятистраничного веб-приложения, также взаимодействующего с теми же десятью таблицами, подключаясь к каждой из них и даже связывая информацию из одной таблицы с информацией из другой таблицы. Затем добавьте к этому обработку изображений (задачу, в решение которой нужно углубиться, чтобы завершить форму пользователя), взаимодействие с Facebook и Twitter, а также разрешение пользователям входить в приложение, и все намного усложнится.

Теперь нам предстоит перейти от размышлений об отдельных формах и сценариях к размышлениям о целых системах. Вы уже вполне готовы приступить к работе с файловой системой, с тем местом, где находятся ваши сценарии, файлы и изображения. Вы готовы приступить к размышлениям не только об отдельной таблице, такой как `users`, но и к мыслям о работе с несколькими таблицами. А как насчет пользовательских функций? Здесь также не о чем волноваться. Вы уже создали две такие функции — `debug_print` и `handle_error`, поэтому с этим проблем не будет.

Но по мере продвижения вперед решения становятся сложнее. Сложности возникают при решении не только вопросов, что делать дальше, но также вопросов, какой из двух-трех возможных способов выполнения той или иной задачи будет лучше. Поэтому приготовьтесь: вы уходите в глубины программирования, в которых свойственно попадать в круговороты критического мышления и философии.

Изображения — это просто файлы

Зияющей дырой в вашей работе остается изображение в профиле. Возможно, вы помните, что работа над пользовательским профилем еще не завершена. Разницу между макетом из главы 6 (рис. 8.1) и тем, где ваш код сейчас (рис. 8.2), определить несложно: она заключается в изображении.

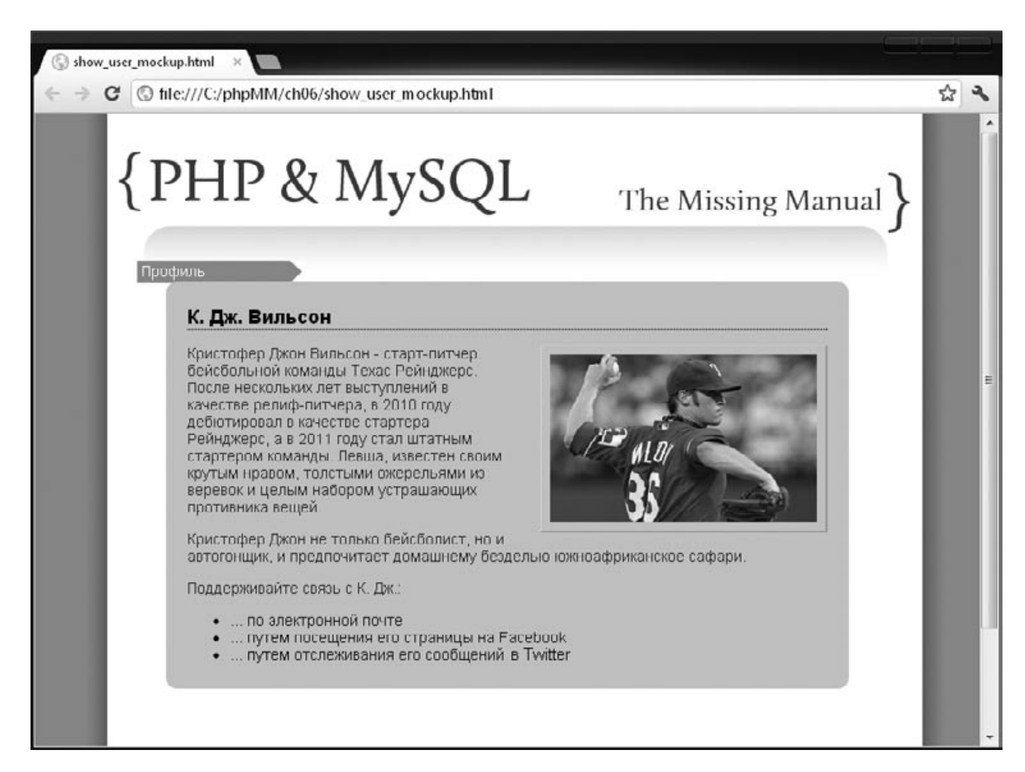


Рис. 8.1. В макете весь интерес сосредоточен на изображении пользователя

Вполне очевидно, что вам требуется изображение для пользователя. С помощью изображения пользователи могут реально персонифицировать страницы своего профиля. (Вам же приходилось видеть аватары в Facebook и Twitter? Людям нравится самовыражаться с помощью картинок в своих профилях!) Но ведь это же совсем несложно, не так ли? Вам уже попадались тысячи тегов `img` в коде HTML:

```
<img src="images/cj_wilson.jpg" class="user_pic" />
```

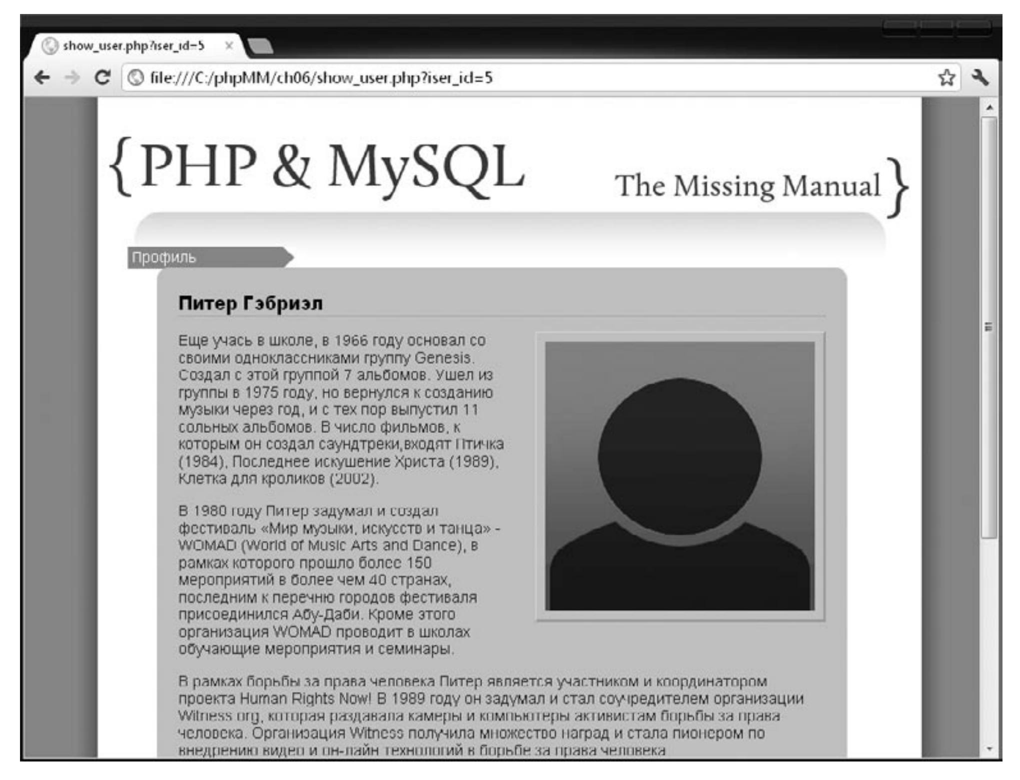


Рис. 8.2. Без изображения страница стала совсем скучной, длинное описание биографии только усугубило положение

Нужно обратить внимание на значение атрибута `src`. Это ссылка на файл, но у вас пока нет никаких файлов изображений. В таблице `users` есть имя пользователя и информация о нем, но на вашем веб-сервере нет изображения, на которое можно указать. Похоже, эта проблема несколько иного сорта. Вам нужна не просто строка текста, вроде Ryan Geyer или @trenspot, а настоящий файл, а затем ссылка на этот файл.

Теперь ваша задача заключается в получении от пользователя чего-то отличающегося от текстовой информации, а затем в решении, что с этой информации делать после ее получения.

К ВАШЕМУ СВЕДЕНИЮ

Файлы, файловые системы и сравнение клиентской и серверной сторон

Возможно, вам впервые понадобится четкое понимание разницы между тем, что такое ваша собственная машина, и тем, что такое сервер.

Вам известно, что такое файл: это просто коллекция битов и байтов, с которой компьютер знает как обращаться. Сценарии HTML, CSS и JavaScript являются в конечном

счете простым текстом: последовательностью символов, которая интерпретируется веб-браузером или PHP-программой. В случае с PHP веб-сервер интерпретирует этот код PHP, превращая его в HTML, CSS и JavaScript для браузера, и затем дает возможность браузеру принять все это. Браузер принимает HTML, CSS и JavaScript в виде либо статического файла, либо данных, возвращенных веб-сервером, который обрабатывал PHP-сценарий, и отображает все это на вашем экране.

А вот изображения являются двоичными данными. Те же самые биты и байты, из которых составлены текстовые файлы, использованы для показа местоположения и цвета пикселов. И для чтения двоичного файла нужен другой тип интерпретации. К счастью, веб-браузеры отлично приспособлены к работе с файлами изображений, имеющими формат JPEG, GIF или PNG, и могут выводить их на экран. Но процесс получения двоичного файла немного другой. Когда пользователи набирают URL вашего веб-приложения в своих браузерах, они запускают вашу программу, которая находится где-то на веб-сервере и доступна через Интернет. Они запускают эту программу, применяя свой веб-браузер, являющийся программой, которая находится на их компьютере. И существует большое различие между тем, что находится на их компьютере, и тем, что располагается на вашем веб-сервере. Поэтому, к примеру, ваш веб-сервер не может забраться в их компьютер и забрать изображения. Чтобы одно из их изображений можно было увидеть в вашей программе, пользователям нужно отправить это изображение на ваш веб-сервер.

Разумеется, большинство пользователей не знает, как отправлять файл, используя программу связи с FTP-сервером. Им это не нужно. Они просто хотят пользоваться программами, а не обучаться работе с множеством таинственных инструментов. Поэтому извлечение файла из их компьютера в вашу файловую систему возлагается на вас. *Файловая система* — всего лишь причудливые слова, означающие систему файлов вашего веб-сервера. Они также могут относиться к вашему пользовательскому компьютеру, то есть клиенту, получающему доступ к вашей программе. А ваша программа работает на сервере. Поэтому все это называется клиент-серверным взаимодействием. Ваша задача заключается в получении файла изображения сервером от клиента. Затем сервер может предоставить вашим PHP-сценариям доступ к этому файлу изображения для его использования в ваших программах и, разумеется, на странице профиля пользователя.

Формы HTML могут готовить почву

В данной ситуации роль HTML куда существеннее простого подыгрывания вашей PHP-программе. Вам нужно убедиться в том, что HTML-форма работает и правильно настроена на помощь пользователям в отправке на сервер их изображений. В этой форме нужно не только выделить пользователю место для выбора изображения, но и настроить процесс, благодаря которому это изображение без всяких сбоев будет отправлено на сервер.

Скопируйте файл `create_user.html` из папки примеров, предназначенных для главы 6, в каталог, в котором сейчас работаете. В том состоянии, в котором мы оставили этот код (см. раздел «Перенаправление и повторное обращение к сценарию,

создающему новых пользователей» главы 6), в нем уже присутствуют некоторые шаги для отправки изображения на сервер:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>
<body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Регистрация </div>

  <div id="content">
    <h1>Вступайте в наш виртуальный клуб</h1>
    <p>Пожалуйста, введите ниже свои данные для связи в Интернете:</p>
    <form action="create_user.php" method="POST"
          enctype="multipart/form-data">
      <fieldset>

        <label for="first_name">Имя:</label>
        <input type="text" name="first_name" size="20" /><br />
        <label for="last_name">Фамилия:</label>
        <input type="text" name="last_name" size="20" /><br />
        <label for="email">Адрес электронной почты:</label>
        <input type="text" name="email" size="50" /><br />
        <label for="facebook_url">URL-адрес в Facebook:</label>
        <input type="text" name="facebook_url" size="50" /><br />
        <label for="twitter_handle">Идентификатор в Twitter:</label>
        <input type="text" name="twitter_handle" size="20" /><br />
        <label for="user_pic">Отправка изображения:</label>
        <input type="file" name="user_pic" size="30" />
        <label for="bio">Биография:</label>

        <textarea name="bio" cols="40" rows=10"></textarea>
      </fieldset>

      <br />

      <fieldset class="center">

        <input type="submit" value="Вступить в клуб" />
        <input type="reset" value="Очистить и начать все сначала" />
      </fieldset>

    </form>

  </div>


  <div id="footer"></div>

 </body>

</html>
```

ПРИМЕЧАНИЕ

Вам также нужно будет изменить действие в форме (form action), чтобы оно отображало, что вы уже не используете каталог scripts/. Этот код HTML можно найти в каталоге примеров ch08/ в загружаемых примерах для данной книги.

Ключевыми составляющими здесь являются атрибут `enctype` в теге `form` и тип ввода (input type) `"file"` для `user_pic`. Эти строки настраивают форму на отправку не только текста, но и двоичного файла изображения.

На рис. 8.3 показано, что страница уже позволяет пользователям выбирать изображение. Но этой странице нужно кое-что еще: ограничение размера изображения. Вам же приходилось получать от друга сообщение электронной почты объемом 22 Мбайт с рисунком кота, раздутым в сто раз по сравнению с его нормальным размером? Нужно избегать подобных случаев в формах. Никаких котов размером в 22 Мбайт. Для любой подходящей картинки профиля вполне достаточно одного-двух мегабайтов.

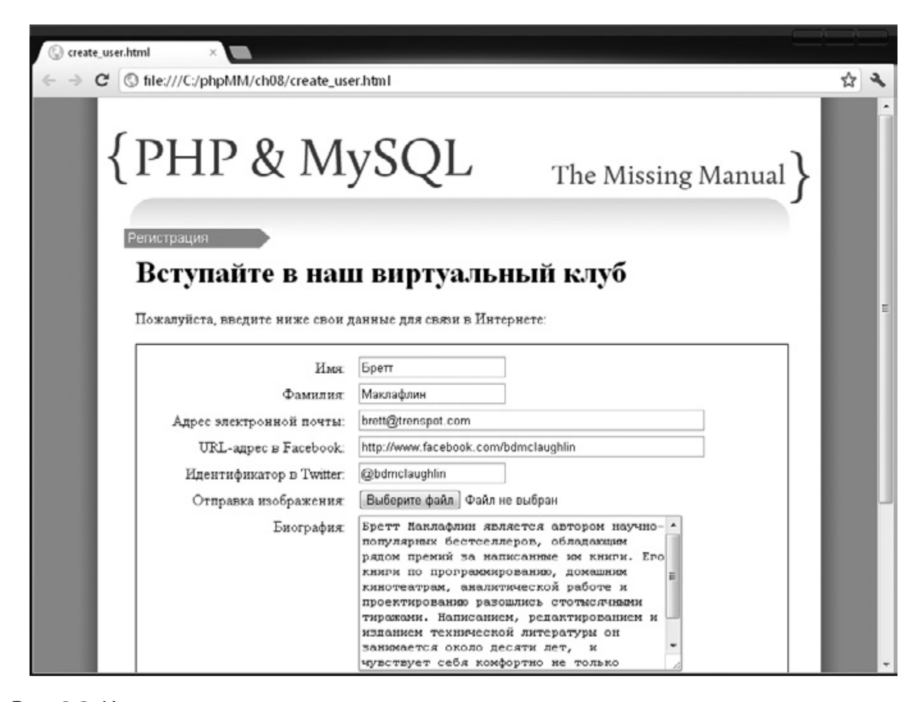


Рис. 8.3. Изменения не видны, но уже есть ограничения, накладываемые на пользователя

ПРИМЕЧАНИЕ

Сокращение Мбайт означает «мегабайт», то есть миллион байтов. Префикс «мега» представляет собой именно это: 1 000 000 единиц чего-то. Чтобы понять, что это за размер, следует отметить, что 20 или 30 страниц документа Word — это около 1 Мбайт. А 20 Мбайт занимает большое изображение.

Вообще, большие изображения необходимы только для сайтов высококачественных фотографий или для сайтов обмена фотографиями, таких как Flickr (www.flickr.com), где по-настоящему ценится масса деталей. Для простой картинки профиля ничего подобного не требуется.

Размер отправляемых файлов можно ограничить путем добавления скрытого элемента ввода и присваивания ему имени "MAX_FILE_SIZE". В качестве значения ему нужно дать максимально разрешенный вами размер отправляемого изображения в байтах. Если вы хотите разрешить использование изображений в 1 Мбайт, значение должно быть равно 1 000 000 байтам. Код HTML, позволяющий отправлять изображения до 2 Мбайт, имеет следующий вид:

```
<input type="hidden" name="MAX_FILE_SIZE" value="2000000" />
<label for="user_pic">Отправка изображения:</label>

<input type="file" name="user_pic" size="30" />
```

ВНИМАНИЕ

Это поле ввода нужно поставить перед полем ввода, имеющим тип "file". Нужно избегать любых комментариев в значении атрибута. Внимательно пересчитайте нули или вы опять вернетесь к шокирующе большим котам. (И для тех, кто этим озабочен, замечу, что при создании данной книги ни один представитель семейства кошачьих не пострадал.)

С этим элементом ввода внешний вид формы нисколько не изменился, но теперь вы уже готовы позволить пользователям отправить изображение, а также готовы что-нибудь сделать с этим изображением (см. рис. 8.3). Попробуйте сделать следующее: выберите изображение, а затем щелкните на кнопке Вступить в клуб. Даже при отсутствии PHP-сценария, ожидающего получения этой информации, вы увидите, что браузер не спеша что-то отправляет. На рис. 8.4 изображена реакция браузера Chrome: побитовая индикация всего происходящего.



Рис. 8.4. Индикатор выполнения

Трудно себе представить, как много всего браузеры делают для вас. Вот пример того, как просто при использовании поля ввода типа «файл» вы получаете индикатор выполнения, сетевое подключение и отправку изображения, и все это бесплатно. Благодаря этому у вас высвобождается время для создания превосходного кода PHP.

Отправка изображения пользователя на ваш сервер

Теперь вы должны получить изображение и что-нибудь с ним сделать. Начнем с копирования старой версии сценария create_user.php в текущий каталог. Сценарий должен иметь следующий вид:

```
<?php

require_once '../scripts/app_config.php';
```

```
require_once '../scripts/database_connection.php';

$first_name = trim($_REQUEST['first_name']);
$last_name = trim($_REQUEST['last_name']);
$email = trim($_REQUEST['email']);
$bio = trim($_REQUEST['bio']);
$facebook_url = str_replace("facebook.org", "facebook.com", trim($_
REQUEST['facebook_url']));
$position = strpos($facebook_url, "facebook.com");
if ($position === false) {
  $facebook_url = "http://www.facebook.com/" . $facebook_url;
}

$twitter_handle = trim($_REQUEST['twitter_handle']);
$twitter_url = "http://www.twitter.com/";
$position = strpos($twitter_handle, "@");
if ($position === false) {
  $twitter_url = $twitter_url . $twitter_handle;
} else {
  $twitter_url = $twitter_url . substr($twitter_handle, $position + 1);
}

$insert_sql = "INSERT INTO users (first_name, last_name, email, " .
                                 "bio, facebook_url, twitter_handle) " .
              "VALUES ('{$first_name}', '{$last_name}', '{$email}', " .
                      "'{$bio}', '{$facebook_url}', '{$twitter_handle}');";

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . mysql_insert_id());
exit();
?>
```

ПРИМЕЧАНИЕ

Чтобы привести сценарий в надлежащее состояние, в него нужно внести несколько изменений. Обновите путь к app_config.php и database_connection.php, а также воспользуйтесь вместо инструкции require инструкцией require_once.

Установка нескольких вспомогательных переменных

Сначала нужно добавить некоторую базовую информацию, используемую для получения файла и для его сохранения. Добавьте в верхнюю часть своей страницы следующие переменные:

```
<?php

require_once '../scripts/app_config.php';
```

```
require_once '../scripts/database_connection.php';

$upload_dir = SITE_ROOT . "uploads/profile_pics/";
$image_fieldname = "user_pic";

$first_name = trim($_REQUEST['first_name']);
// Другие переменные

// Получение информации запроса

// Вставка в MySQL
?>
```

Вполне обычный код. Здесь используется та же самая константа SITE_ROOT, которая была определена в app_config.php. Благодаря ее применению вы можете определить каталог, в котором хотите сохранить отправленные файлы. Создайте этот каталог на вашем веб-сервере, воспользовавшись программой Terminal, окном командной строки или FTP-программой. Если же константа SITE_ROOT указывает просто на /, создайте каталог /uploads/profile_pics. Если константа SITE_ROOT указывает на yellowtagmedia_com/phpMM, нужно создать каталог yellowtagmedia_com/phpMM/uploads/profile_pics.

Теперь необходимо добавить следующий массив потенциальных ошибок:

```
$upload_dir = SITE_ROOT . "uploads/";
$image_fieldname = "user_pic";

// Потенциальные PHP-ошибки отправки файлов
$php_errors = array(1 => 'Превышен макс. размер файла, указанный в php.ini',
                2 => 'Превышен макс. размер файла, указанный в форме HTML',
                3 => 'Была отправлена только часть файла',
                4 => 'Файл для отправки не был выбран.');
```

Вы уже использовали массивы, но здесь делается нечто новое. Мы создаем новый массив с помощью ключевого слова array, а затем определяем значения, попадающие в массив.

Поскольку массив по сути является списком значений, можно было бы просто создать следующий код:

```
// Потенциальные PHP-ошибки отправки файлов
$php_errors = array('Превышен макс. размер файла, указанный в php.ini',
                    'Превышен макс. размер файла, указанный в форме HTML',
                    'Была отправлена только часть файла',
                    'Файл для отправки не был выбран.');
```

В этом массиве каждое значение автоматически нумеруется, начиная с нуля. Следовательно, элемент $php_errors[0] будет, к примеру, иметь значение 'Превышен макс. размер файла, указанный в php.ini'.

ВНИМАНИЕ

Следует помнить, что почти во всех языках программирования придется иметь дело с отсчетом, начинающимся с нуля, а не с единицы (см. врезку «Под капотом. Языкам программирования нравится начинать все с нуля» раздела «Поиск в тексте» главы 2).

А что означают эти числа и забавные стрелки (=>)? Дело в том, что массивы PHP являются ассоциативными. Именно поэтому вы можете использовать, например, такие выражения, как $_REQUEST['user_pic']. Массив $_REQUEST не только содержит значения, но и имеет связи между такими значениями (обычно информацию в HTML-форме) и именами полей, в которых они появляются.

Это можно представить как проецирование имени поля user_pic на значение этого поля, например на profile_pic.jpg, определяемое с помощью следующего выражения:

```
$_REQUEST = array('user_pic' => 'profile_pic.jpg');
```

ПРИМЕЧАНИЕ

В PHP обработка информации происходит намного сложнее, чем может показаться на первый взгляд. Это позволяет определять для формы любое нужное вам поле любого необходимого вам типа с любым нужным вам именем, PHP со всем этим справится. В конечном итоге в процессе обработки будет создан ассоциативный массив с именами полей, связанными со значениями этих полей, или, иначе говоря, с именами, которые спроецированы на значения.

Вернемся к вашему массиву PHP-ошибок:

```
// Потенциальные PHP-ошибки отправки файлов
$php_errors = array(1 => 'Превышен макс. размер файла, указанный в php.ini',
                2 => 'Превышен макс. размер файла, указанный в форме HTML',
                3 => 'Была отправлена только часть файла',
                4 => 'Файл для отправки не был выбран.');
```

Этот массив принимает нумерацию из ваших рук, не позволяя PHP самостоятельно определить номера элементов. Таким образом, $php_errors[1] теперь имеет значение 'Превышен макс. размер файла, указанный в php.ini', и интерпретатору PHP не позволено применить присваивание с нулевой базой к этой строке и сделать ее значением элемента $php_errors[0].

Но зачем вам вносить путаницу в PHP-нумерацию? В целом это не самая удачная затея, поскольку вы вмешиваетесь в поведение, ожидаемое всеми PHP-программистами. Но в данном случае для этого есть довольно веская причина.

Ведь PHP не только предоставляет вам массив $_REQUEST. Когда ведется работа с файлами, итерпретатор этого языка создает массив $_FILES. И в этом массиве точно так же, как и в $_REQUEST, ключом служит имя вашего поля. Таким образом, элемент массива $_FILES[$image_fieldname] будет связан с отправляемым из вашей формы изображением. (Вспомните, что переменная $image_fieldname определена в самом начале сценария create_user.php.)

Но это еще не все. Элемент $_FILES[$image_fieldname] сам по себе является массивом, содержащим информацию об отправленном файле и о любых ошибках, которые могут случиться в процессе отправки. Частицей такой информации является элемент $_FILES[$image_fieldname]['error']. Это поле возвращает число 0 для ситуации «Все прошло удачно», и число, отличающееся от нуля, в случае возникновения проблем. Текст в массиве показывает, что означает каждое ненулевое число:

```
1 => 'Превышен макс. размер файла, указанный в php.ini'
2 => 'Превышен макс. размер файла, указанный в форме HTML'
3 => 'Была отправлена только часть файла'
4 => 'Файл для отправки не был выбран.'
```

Теперь перенумерованный массив $php_errors имеет смысл: вы получаете карту кодов ошибок, которую может вернуть элемент $_FILES[$image_fieldname]['error'], и сопутствующие этим кодам описания ошибок в удобочитаемом виде.

Итак, теперь у вас есть вся необходимая информация. Настало время приступить к ее использованию.

Были ли ошибки при отправке файла?

Вы знаете, что делать: проверить именно эту частицу массива $_FILES и посмотреть, не случилась ли ошибка. Если значение отлично от нуля, что-то дало сбой и вам нужно обработать проблему. К счастью, именно для этого у вас есть удобная функция handle_error.

```
<?php
// Затребование сервисных сценариев
// Установка значений переменных
// Получение всего из формы кроме изображения

// Проверка отсутствия ошибки при отправке изображения
($_FILES[$image_fieldname]['error'] == 0)
  or handle_error("сервер не может получить выбранное вами изображение.",
                $php_errors($_FILES[$image_fieldname]['error']));

// Взаимодействие с MySQL
// Перенаправление на show_error.php
?>
```

Если поле ошибки ($_FILES[$image_fieldname]['error']) имеет значение 0, значит, все прошло успешно и нужно продолжить выполнение сценария. Если оно имеет значение, отличное от нуля, пользователю нужно показать ошибку, а затем воспользоваться кодом этой ошибки, чтобы посмотреть в вашем ассоциативном массиве $php_errors конкретную причину проблемы и передать ее для вывода на экран, если включен режим отладки.

ПРИМЕЧАНИЕ

Теперь в самый раз будет проверить app_config.php и убедиться, что значением константы DEBUG_MODE является true.

Здесь также имеется небольшая особенность, на которую вы, возможно, не обратили внимания: это по сути инструкция `if` без ключевого слова `if`. Интерпретатор PHP вычислит значение этой строки кода:

```
($_FILES[$image_fieldname]['error'] == 0)
```

И если это значение будет равно `true`, продолжит выполнение сценария. Если значение строки не будет равно `true`, интерпретатор PHP выполнит часть `or` этого кода, которая находится на следующей строке, в данном случае это будет вызов функции `handle_error`.

Эта строка по существу похожа на следующий код:

```
if ($_FILES[$image_fieldname]['error'] != 0) {
  handle_error("сервер не может получить выбранное вами изображение.",
               $php_errors[$_FILES[$image_fieldname]['error']]);
}
```

ВНИМАНИЕ

Здесь нужно очень внимательно следить за расстановкой квадратных и круглых скобок. Их нетрудно перемешать и запутаться, в результате чего возникнет трудно выявляемая ошибка.

Но этот код несколько длиннее, в то время как код без `if` немного более понятен. Здесь во благо любое даже самое незначительное упрощение, поэтому данный привлекательный трюк вы должны добавить в свой арсенал приемов работы с PHP.

Сейчас вы можете проверить код в действии. Вызовите страницу `create_user.html` и найдите файл изображения, размер которого превышает 2 Мбайт. Поищите фотографию в iPhoto или среди снимков, извлеченных непосредственно из вашей камеры. Выберите подходящее изображение, а затем отправьте свою форму. Ответ должен быть примерно таким, как показано на рис. 8.5.

Таким образом, мы рассмотрели пример замечательной ситуации, когда вложенный ранее большой объем труда позднее приносит свои плоды. Вместо того чтобы копаться в коде или даже создавать какие-то кодовые фрагменты на PHP, вы получаете возможность быстро передать ошибку своей функции `handle_error` и получить от нее весьма элегантный ответ. А теперь умножьте все это на несколько сотен (или тысяч?) раз использования `handle_error`, и вы начнете понимать ценность обладания столь полезной функцией, написанной в самом начале вашей PHP-карьеры.

ПРИМЕЧАНИЕ

Вы могли заметить, что даже притом, что изображение было забраковано, браузер все равно отправляет его, независимо от того, насколько оно большое по объему или какой максимальный размер файла вами установлен. Это происходит потому, что оценка размера происходит только после того, как изображение отправлено. С этим недоразумением нельзя разобраться с помощью PHP, его решение должно быть возложено на ваш ленивый браузер.

Эта страница является результатом того, что код нашел код ошибки. Код был сопоставлен с ключом массива `$php_errors`, и в данном случае изображение было больше, чем разрешал код HTML.

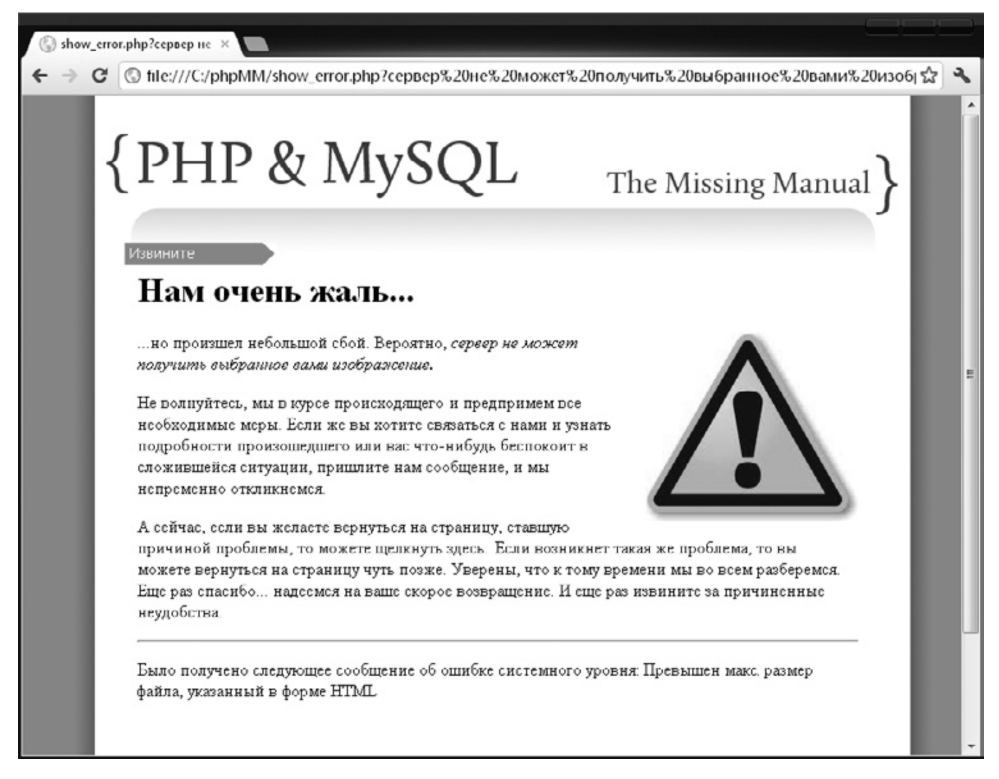


Рис. 8.5. Сообщение об ошибке загрузки изображения

А это действительно отправляемый файл?

Следующее наше действие почему-то не рассмотрено в основной массе инструкций и книг по PHP, но оно тем не менее играет весьма важную роль. Именно сейчас, независимо от того, есть у вас реальный файл или его нет, ваша программа должна поработать с его *именем*. И это имя полностью контролируется тем, что пользователи поместили в свое поле ввода файла. Поэтому, если они схитрили, задумали что-то недоброе и поступили нечестно, они могли поместить в поле имени файла данные, отправляющие файл на их систему. И к тому же может случиться, что это будет один из тех специальных управляющих файлов, которые имеются на веб-серверах, например файл паролей пользователей. (В его роли обычно выступает файл `/etc/passwd`.)

Может сложиться ощущение, что здесь придется воспользоваться регулярными выражениями, чтобы проверить все разновидности предполагаемых символов имен файлов, но есть более легкий путь. PHP предоставляет вам функцию под названием `is_uploaded_file`, которая гарантирует, что заданное имя имеет отношение к файлу, который отправлен на сервер по протоколу HTTP (используемому веб-браузерами и HTML-формами). Иными словами, если предоставленное имя будет указывать на файл на вашем веб-сервере, эта функция вернет `false` и вы узнаете, что происходит что-то подозрительное.

Итак, нужно сделать что-либо подобное:

```
// Убеждаемся, что при отправке изображения не произошла ошибка

// Является ли этот файл результатом нормальной отправки?
is_uploaded_file($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы попытались совершить безнравственный поступок. Позор!",
                  "Запрос на отправку: файл назывался " .
                  "'{$_FILES[$image_fieldname]['tmp_name']}'");

// Взаимодействие с MySQL
```

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Значение сна и передышки

Любой хороший программист расскажет вам истории, касающиеся по крайней мере нескольких ночных бдений за компьютером. И вероятнее всего, эти истории будут преподнесены в позитивных тонах, полных победных реляций и эмоций. Но на самом деле усталость снижает активность мозга, и ни один программист не будет работать столь же эффективно, поспав вместо шести всего два часа.

Лучше работать на свежую голову. Вы уже получили до этой главы большой объем информации о PHP-программировании. Но далее объем будет только увеличиваться. Возможно, некоторые фрагменты текста придется читать дважды, а в некоторых примерах кода будут представлены не одна-две новинки, а три, четыре или пять.

При переутомлении от вашего упорства никто не выиграет. Отдохните пару часов, покатайтесь на велосипеде, пробегите пару-другую километров или просто отложите изучение PHP до следующего утра. И вы удивитесь тому, насколько понятнее все станет после небольшого отдыха от программирования.

В этом коде используется еще одно свойство (`$_FILES[$image_fieldname]`): временное имя файла. Благодаря ему вы получаете имя файла на данный момент и убеждаетесь в том, что это отправленный на сервер файл.

Но здесь есть еще одна проблема: `is_uploaded_file` выдает ошибку, если файл не был отправлен. Вроде бы неплохо, за исключением того, что вы уже приложили немало усилий для обработки ошибок на свой лад. Вам не нужно, чтобы `is_uploaded_file` генерировала ошибку. Необходимо лишь, чтобы она при возникновении проблемы возвращала значение.

Можно заставить PHP запустить функцию и подавить выдачу ошибок, поставив непосредственно перед именем функции символ `@` (подробнее об этом читайте в следующей врезке), что нам собственно и нужно в данной ситуации:

```
// Является ли этот файл результатом нормальной отправки?
@is_uploaded_file($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы попытались совершить безнравственный поступок. Позор!",
                  "Запрос на отправку: файл назывался " .
                  "'{$_FILES[$image_fieldname]['tmp_name']}'");
```

Теперь функция запускается. Если возникнет проблема, ее возьмет на себя функция `handle_error` и ваш сценарий уже не станет самостоятельно выдавать какую-нибудь непонятную ошибку. Это становится еще одним препятствием для взломщика сайта, что снова оправдывает такое добавление и закрывает брешь в защите.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Подавление выдачи ошибок делается на ваш собственный страх и риск

Наверное, в PHP нет более интригующего оператора, чем @. Одним нажатием клавиши можно обойти все проблемы, которые могут возникать при вводе пользователем неверных данных, или при наличии в SQL-запросе неверного столбца, или даже просто при неправильно сформированном URL. Выполнение вашего кода может продолжиться, и вам не придется проверять, не возникло ли какой-нибудь ошибки, которую могут допустить ваши пользователи, вы и ваш код.

Но оператор @ — это настоящая атомная бомба, которая только и ждет того, чтобы превратить ваш код в тлеющие головешки. При его частом использовании вы быстро столкнетесь с потенциальными проблемами. Вы никогда не сможете понять, с чем связана проблема: с действиями вашего пользователя, с вашими действиями или с настоящей ошибкой, требующей исправления.

Независимо от того, чем вызвана ошибка, если она обойдена с помощью оператора @, стало быть, она узаконена. Возьмите за правило: при использовании @ (как в следующей строке кода) применяйте с ним в паре инструкцию `or` и код для явной обработки ошибки. Такая дисциплина существенно улучшит ситуацию.

Но есть одно исключение. (А как без них!) Это исключение связано с крупными, популярными или важными веб-сайтами. На таких сайтах часто используется оператор @, поскольку они просто не могут дать сбой и остановить свою работу. В подобных случаях приходится прибегать к некому составному решению. С одной стороны, использовать @, но затем применять вместе с ним инструкцию `or` с кодом, выполняемым в случае выставления флажка отладки, как описывалось в подразделе «Сейчас вы меня видите, а сейчас нет» раздела «Добавление отладки к приложению» главы 7. Таким образом, в «нормальном» режиме все выполняется без выдачи массы ошибок (или, возможно, только лишь с их регистрацией). А затем путем переключения в режим отладки вы начинаете видеть, что же происходит на самом деле, и можете отследить и устранить возникающие проблемы.

Является ли отправленный файл изображением?

Итак, у вас есть отправленный на сервер файл и вы знаете, что он не из разряда подделок с именем, указывающим на какой-нибудь защищенный файл в файловой системе вашего сервера. Но вам предстоит сделать еще один шаг: убедиться в том, что это файл изображения. Ничто не мешает пользователю случайно отправить документ Word, а также ничто не препятствует отправке злоумышленником какого-нибудь файла с кодом JavaScript или исполняемого файла.

К счастью, PHP позволяет довольно просто проверять принадлежность файлов к изображениям. Для этого нужно воспользоваться функцией getimagesize, которая производит проверку размера заданного файла изображения. И что лучше всего — эта функция выдает ошибку, если заданный файл не является файлом изображения. Отлично!

```
// Действительно ли это изображение?
@getimagesize($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы выбрали файл для своего фото, " .
                  "который не является изображением.",
                  "{$_FILES[$image_fieldname]['tmp_name']} " .
                  "не является файлом изображения.");
```

Перемещение файла в постоянное место хранения

Вы уже почти подошли к финишу. Произошла правильная отправка файла по протоколу HTTP, и этот файл является изображением. Осталось только переместить это изображение из временного места хранения, которое используется веб-серверами для отправленных на них файлов, в какое-нибудь постоянное место. Здесь пригодится ваша переменная из далекого прошлого:

```
$upload_dir = SITE_ROOT . "uploads/profile_pics/";
```

ПРИМЕЧАНИЕ

Если этот каталог еще не создан, то сейчас самое время сделать это.

На данный момент важно знать, что произошло с файлом, отправленным вашим пользователем. Когда сервер получает этот файл, он применяет для него место, заранее определенное конфигурацией. Он также, скорее всего, воспользуется именем, не совпадающим с исходным именем пользовательского файла. Иногда имя полностью изменяется, а иногда впереди или сзади него что-нибудь добавляется.

Кроме того, файл не находится в том месте, в котором вам хотелось бы его оставить. Зачастую он остается в каком-нибудь временном хранилище, и это хранилище довольно часто полностью очищается. Поэтому вам нужно не только присвоить файлу имя, но и переместить его куда-нибудь в место постоянного хранения. Именно для этого у вас есть переменная $upload_dir.

Существует множество различных подходов к присваиванию имени. Вы можете придумать что-нибудь связанное с пользователем, отправившим файл на сервер, но зачастую проще всего дать файлу уникальное числовое имя. И самым простым способом для этого служит получение текущего времени и создание имени файла на его основе — это практически надежный путь получения уникального имени файла.

ПРИМЕЧАНИЕ

Посмотрите на имена изображений на таких сайтах, как Flickr или Facebook. Пока пользователи не переименуют свои изображения, их имена зачастую представляют собой строку, состоящую из букв и цифр.

Итак, вам нужно получить уникальное имя, а затем вы сможете окончательно переместить файл из его текущего местоположения на постоянное место.

Сначала определим имя для изображения, которому предстоит скорое перемещение:

```
// Присваивание файлу уникального имени
$now = time();
while (file_exists($upload_filename = $upload_dir . $now .
                                      '-' .
                                      $_FILES[$image_fieldname]['name'])) {
    $now++;
}
```

Рассмотрим действия, которые нужно выполнить, пошагово.

1. Создайте новую переменную по имени $now и присвойте ей значение текущего времени, используя PHP-функцию time.
2. Запустите цикл while. Пока определенные условия будут соблюдаться и возвращать true, цикл будет продолжаться. Как только условие будет иметь значение, отличное от true, цикл завершится.
3. Присвойте прямо в условии цикла while значение переменной $upload_filename: это будет значение переменной $upload_dir плюс текущее время, а затем укажите дефис (-) и в конце имя исходного файла. Получится комбинация из уникальной части (времени) и исходного имени пользовательского файла (которое находится в $_FILES[$image_fieldname]['name']).
4. Затем для завершения условия цикла while передайте это вычисленное имя файла функции file_exists. Если такой файл существует, запускается цикл while. Если нет, вы получаете уникальное имя файла и цикл не запускается (или больше не запускается, если он уже запускался).
5. Внутри цикла нужно придумать способ изменения имени файла. Поскольку цикл while будет запускаться, только если будет получено уже используемое имя файла, нужно просто увеличить значение переменной $now и повторить попытку.

И в этом проявляется вся прелесть PHP: все это можно сделать всего лишь в нескольких строчках кода. Когда данный код завершит свое выполнение, вы получите уникальное имя для пользовательского файла.

Теперь переместите файл из его старого временного местоположения в постоянное место:

```
// И наконец, перемещение файла на его постоянное место
@move_uploaded_file($_FILES[$image_fieldname]['tmp_name'], $upload_filename)
  or handle_error("возникла проблема сохранения вашего изображения " .
                  "в его постоянном месте.",
                  "ошибка, связанная с правами доступа при перемещении " .
                  "файла в {$upload_filename}");
```

Была проделана большая работа, но в итоге файл оказался в постоянном месте хранения и вы знаете, что он является настоящим изображением. Попробуйте все это в работе. Зайдите на страницу, создаваемую сценарием create_user.php, выберите изображение на своем жестком диске, подпадающее под ограничения размера, и отправьте его на сервер. Затем из веб-браузера перейдите к соответствующему каталогу. Если у вас есть разрешение на просмотр каталогов на вашем веб-сервере, то вы увидите изображение, похожее на то, что показано на рис. 8.6. (Но вообще, если вам удалось увидеть файлы таким образом, особо радоваться не стоит. Чаще всего это означает, что по структуре каталогов вашего веб-сервера может бродить кто угодно. И хотя для отладки это весьма удобно, оставлять такой режим включенным вам вряд ли захочется. В связи с этим может потребоваться послать сообщение электронной почты, позвонить провайдеру вашего веб-сервера или в хостинг-компанию, чтобы попросить отключить просмотр каталогов через Интернет.)

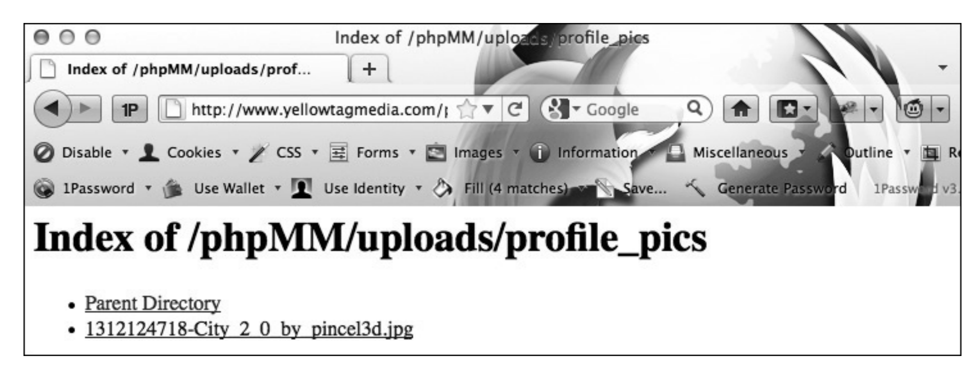


Рис. 8.6. Каталог с изображениями на веб-сервере

Если вы не увидите каталог или получите сообщение о том, что в просмотре содержимого каталогов вам отказано, не нужно расстраиваться. Это обычная практика работы веб-серверов, и она совсем еще не означает, что что-то пошло не так.

Вместо этого откройте create_user.php и внесите в него два изменения. Сначала добавьте в него команду echo для вывода местоположения вашего файла:

```
// Присваивание файлу уникального имени
$now = time();
while (file_exists($upload_filename = $upload_dir . $now .
                                      '-' .
                                      $_FILES[$image_fieldname]['name'])) {
    $now++;
}

echo $upload_filename;
echo "<br />";
echo $_FILES[$image_fieldname]['tmp_name'];
```

А затем нужно закомментировать перенаправление, чтобы результаты выполнения команды echo можно было увидеть:

```
// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
//header("Location: show_user.php?user_id=" . mysql_insert_id());
```

Отправьте изображение на сервер еще раз, и вы получите путь, выведенный командой echo, если, конечно, все пройдет успешно. Ожидаемый вывод показан на рис. 8.7.

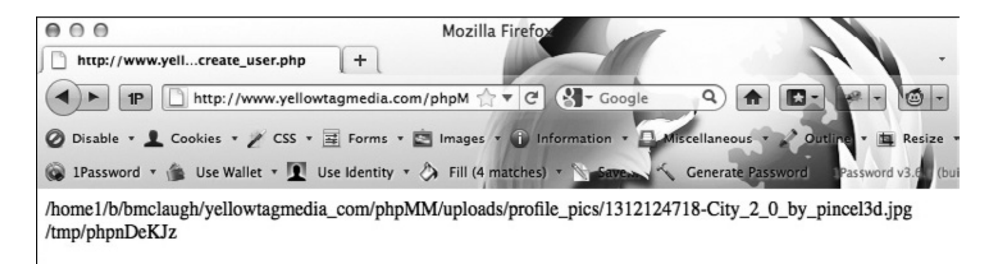


Рис. 8.7. Вывод команды echo

Теперь (наконец-то!) вы можете поместить путевое имя этого файла в адресную строку, указав сначала имя своего домена, и получить великолепное изображение с машины вашего пользователя на вашем веб-сервере (рис. 8.8).



Рис. 8.8. Изображение попало в нужное место

Сохранение местоположения изображения в базе данных

Время потрачено не зря, и теперь вы наконец-то готовы сохранить это изображение или по крайней мере его местонахождение в своей базе данных. На данный момент у вас уже составлен следующий запрос:

```
$insert_sql = "INSERT INTO users (first_name, last_name, email, " .
                           "bio, facebook_url, twitter_handle) " .
              "VALUES ('{$first_name}', '{$last_name}', '{$email}', " .
                      " '{$bio}', " . '{$facebook_url}', " .
                      "'{$twitter_handle}');";

// Вставка пользователя в базу данных
mysql_query($insert_sql);
```

Создание нового столбца в таблице базы данных

Теперь нужно добавить столбец, в котором можно будет хранить местоположение изображения. Этим займется еще одна команда ALTER, с которой вы уже знакомы:

```
ALTER TABLE users
        ADD user_pic_path varchar(200);
```

Запустите эту инструкцию. Чтобы убедиться в ее успешном завершении, может понадобиться получить описание вашей таблицы users с помощью инструкции DESCRIBE:

```
mysql> describe users;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| user_id        | int(11)      |      | PRI | NULL    | auto_increment |
| first_name     | varchar(20)  |      |     |         |                |
| last_name      | varchar(30)  |      |     |         |                |
| email          | varchar(50)  |      |     |         |                |
| facebook_url   | varchar(100) | YES  |     | NULL    |                |
| twitter_handle | varchar(20)  | YES  |     | NULL    |                |
| bio            | text         | YES  |     | NULL    |                |
| user_pic_path  | varchar(200) | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

Данное поле user_pic_path — это всего лишь текстовый столбец, поскольку в нем хранится не само изображение, а только путь к нему.

ПРИМЕЧАНИЕ

Если вас охватило любопытство насчет того, как можно хранить в базе данных само изображение, то вы можете его удовлетворить в следующем разделе, в котором будет обсуждаться вопрос, насколько хороша данная затея.

Вставка пути к изображению в вашу таблицу

Теперь можно легко и просто обновить запрос INSERT:

```
$insert_sql = "INSERT INTO users (first_name, last_name, email, " .
                         "bio, facebook_url, twitter_handle, " .
                         "user_profile_pic) " .
              "VALUES ('{$first_name}', '{$last_name}', '{$email}', " .
                      " '{$bio}', " . '{$facebook_url}', " .
                      "'{$twitter_handle}', '{$upload_filename}');";

// Вставка пользователя в базу данных
mysql_query($insert_sql);
```

Теперь все пошло значительно быстрее. Когда проделана такая работа, добавление нового столбца становится довольно легкой задачей. Но перед возвращением в глубины вашего кода HTML осталось сделать еще одну вещь.

Проверка сделанного

Созданный выше запрос должен работать, но как это проверить? Если бы вы были только лишь PHP-программистом, то вам пришлось бы попробовать выполнить данный код, а затем либо написать новый сценарий для выборки данных из таблицы users, либо вернуться непосредственно к сценарию show_user.php. Но зачем все эти сложности? Вы ведь знаете SQL и приемы взаимодействия с MySQL.

Сначала создайте нового пользователя и возьмите для него ранее неприменявшееся имя. Затем вернитесь к окну командной строки SQL и самостоятельно проверьте результаты своей работы. С помощью инструкции SELECT выберите только что вставленного пользователя, сконцентрировавшись на пути к его изображению:

```
SELECT user_pic_path
  FROM users
 WHERE last_name = 'Roday';
```

Вы должны увидеть нечто подобное:

```
mysql> select user_pic_path from users where last_name = 'Roday';
+-----------------------------------+
| user_pic_path                     |
+-----------------------------------+
| /yellowtagmedia_com/phpMM/uploads/profile_pics/1312127661-City_2_0_by_pince-l3d.jpg |
+-----------------------------------+
1 row in set (0.00 sec)
```

Превосходно. Изображение находится на вашем веб-сервере, а теперь вы получили надежно сохраненный в вашей базе данных путь к этому изображению. Сейчас вы готовы показать пользователям их замечательное изображение, которое они сами и выбрали.

Если есть какие-то вопросы, можно просмотреть показанную ниже полную версию сценария create_user.php. Он претерпел множество всяких добавлений, поэтому нужно просто убедиться в том, что все на месте:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';

$upload_dir = SITE_ROOT . "uploads/profile_pics/";
$image_fieldname = "user_pic";

// Потенциальные PHP-ошибки отправки файлов
$php_errors = array(1 => 'Превышен макс. размер файла, указанный в php.ini',
                2 => 'Превышен макс. размер файла, указанный в форме HTML',
                3 => 'Была отправлена только часть файла',
                4 => 'Файл для отправки не был выбран.');

$first_name = trim($_REQUEST['first_name']);
$last_name = trim($_REQUEST['last_name']);
$email = trim($_REQUEST['email']);
$bio = trim($_REQUEST['bio']);
$facebook_url = str_replace("facebook.org", "facebook.com", trim($_
REQUEST['facebook_url']));
$position = strpos($facebook_url, "facebook.com");
if ($position === false) {
  $facebook_url = "http://www.facebook.com/" . $facebook_url;
}

$twitter_handle = trim($_REQUEST['twitter_handle']);
$twitter_url = "http://www.twitter.com/";
$position = strpos($twitter_handle, "@");
if ($position === false) {
  $twitter_url = $twitter_url . $twitter_handle;
} else {
  $twitter_url = $twitter_url .
                 substr($twitter_handle, $position + 1);
}

// Проверка отсутствия ошибки при отправке изображения
($_FILES[$image_fieldname]['error'] == 0)
  or handle_error("сервер не может получить выбранное вами изображение.",
                  $php_errors[$_FILES[$image_fieldname]['error']]);

// Является ли этот файл результатом нормальной отправки?
@is_uploaded_file($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы попытались совершить безнравственный поступок. Позор!",
```

```
                  "Запрос на отправку: файл назывался " .
                  "'{$_FILES[$image_fieldname]['tmp_name']}'");

// Действительно ли это изображение?
@getimagesize($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы выбрали файл для своего фото, " .
                  "который не является изображением.",
                  "{$_FILES[$image_fieldname]['tmp_name']} " .
                  "не является настоящим файлом изображения.");

// Присваивание файлу уникального имени
$now = time();
while (file_exists($upload_filename = $upload_dir . $now .
                                     '-' .
                   $_FILES[$image_fieldname]['name'])) {
    $now++;
}

// И наконец, перемещение файла на его постоянное место
@move_uploaded_file($_FILES[$image_fieldname]['tmp_name'],
                    $upload_filename)
  or handle_error("возникла проблема сохранения вашего изображения " .
                  "в его постоянном месте.",
                  "ошибка, связанная с правами доступа при перемещении " .
                  "файла в {$upload_filename}");

$insert_sql = "INSERT INTO users (first_name, last_name, email, " .
              "bio, facebook_url, twitter_handle, user_pic_path) " .
              "VALUES ('{$first_name}', '{$last_name}', '{$email}', " .
              "'{$bio}', '{$facebook_url}', '{$twitter_handle}', " .
              "'{$upload_filename}');";

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . mysql_insert_id());
exit();
?>
```

Изображения, предназначенные для просмотра

И вот, наконец, настало время показать вашим пользователям плоды всей вашей нелегкой работы. Им, наверное, невдомек, как долго и упорно вы трудились над выводом одного-единственного изображения и над защитой всей другой информации в процессе этого вывода, но иногда вполне достаточно того, что вы сами об этом знаете.

Поместите копию файла show_user.php рядом с файлами create_user.html и create_user.php. Вам нужно обновить show_user.php для выбора пути к картинке пользователя из таблицы users и последующего отображения этого изображения.

ПРИМЕЧАНИЕ

Как и при обновлении всех остальных сценариев не забудьте заменить инструкцию require инструкцией require_once, включить ссылку на app_config.php и обновить пути, показав, что вы уже не используете каталоги scripts/, относящиеся к той или иной главе. В любых сценариях, имеющих HTML, таких как show_user.php, вы также должны проверить пути к таблицам CSS и внешние ссылки JavaScript.

Выбор изображения с помощью инструкции SELECT и вывод его на экран

Этот шаг представляется не самым сложным. Начнем с того, что вы уже применяли инструкцию SELECT, которая извлекала всю информацию о конкретном пользователе:

```
// Создание инструкции SELECT
$select_query = "SELECT * FROM users WHERE user_id = " . $user_id;
```

Затем вы можете просто добавить строку, извлекающую путь к изображению в код, с помощью которого вы уже извлекали информацию из результатов запуска SQL-инструкции INSERT:

```
if ($result) {
  $row = mysql_fetch_array($result);
  $first_name     = $row['first_name'];
  $last_name      = $row['last_name'];
  $bio            = preg_replace("/[\r\n]+/", "</p><p>", $row['bio']);
  $email          = $row['email'];
  $facebook_url   = $row['facebook_url'];
  $twitter_handle = $row['twitter_handle'];
  $user_image     = $row['user_pic_path'];

  // Превращение $twitter_handle в URL
  $twitter_url = "http://www.twitter.com/" .
                 substr($twitter_handle, $position + 1);

} else {
  handle_error("возникла проблема с поиском вашей " .
               "информации на нашей системе.",
               "Ошибка обнаружения пользователя с ID {$user_id}");
}
```

ПРИМЕЧАНИЕ

Воспользуйтесь этой возможностью для перехода от использования die в блоке else вашей инструкции if к применению более эффективной функции handle_error.

Убедитесь в том, что вы полностью удалили следующий старый код:

```
// Для последующего добавления
$user_image = "../../images/missing_user.png";
```

И наконец, у вас уже есть место в HTML-коде этого сценария, которое ссылается на переменную $user_image:

```
<div id="content">
  <div class="user_profile">
    <h1><?php echo "{$first_name} {$last_name}"; ?></h1>
    <p><img src="<?php echo $user_image; ?>" class="user_pic" />
  <!-- и т. д... -->
```

Испытайте этот код в работе. Вызовите страницу show_user.php с существующим пользовательским ID в адресной строке своего браузера или создайте нового пользователя с картинкой и дайте возможность сценарию create_user.php осуществить перенаправление. Вы должны увидеть нечто похожее на рис. 8.9. Вряд ли вы надеялись на рев фанфар и танцующих ангелов.

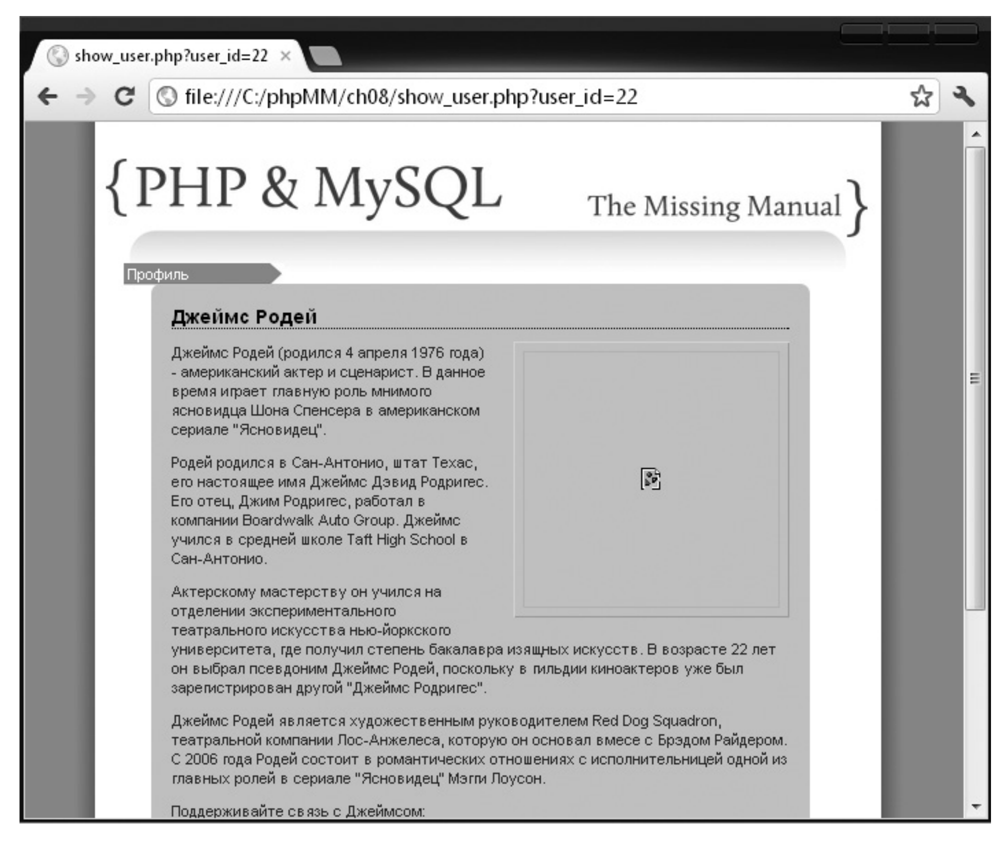


Рис. 8.9. Результат выполнения кода

При получении неожиданного результата, например отсутствия изображения, начните с просмотра исходного кода (через пункт меню, вызывающий просмотр кода, или путем щелчка правой кнопкой и выбора аналогичного пункта контекстного меню) или воспользуйтесь дополнительным модулем вроде Firebug для изу-

чения проблемных элементов. Это практически всегда будет хорошим первым шагом по выявлению причины случившегося.

При просмотре исходного кода страницы и определении используемого путевого имени изображения вы, скорее всего, увидите картину, похожую на ту, что изображена на рис. 8.10.

```
show_user_.php?user_id=22
view-source:file:///C:/phpMM/ch08/show_user_.php?user_id=22

<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: Missing Manual</h1></div>
  <div id="example">Профиль</div>
  <div id="content">
    <div class="user_profile">
      <h1>Джеймс Родей</h1>
      <img src="/home1/b/bmclaugh/yellowtagmedia_com/phpMM/uploads/profile_pics/1312128274-james_roday.jpg" class="user_pic"
      <p>Джеймс Родей (родился 4 апреля 1976 года) - американский актер и сценарист. В данное время играет
```

Рис. 8.10. Исходный код страницы

В соответствии с данным исходным кодом HTML у элемента `img` имеется правильное абсолютное путевое имя изображения. Но так ли путевое имя должно выглядеть в HTML-коде страниц? Какое отношение имеет абсолютный путь в файловой системе к пути на веб-сервере?

Проверка на принадлежность к изображению у вас проводится на более ранней стадии. Поэтому вы знаете, что путь к изображению не является источником проблем... Или все же является?

Преобразование путей файловой системы в URL-адреса

То, чем вы располагаете, является путем в файловой системе вашего веб-браузера, а вам нужен путь, распознаваемый вашим веб-сервером. У каждого веб-сервера есть *корневой каталог документов*. Это каталог, в который помещаются файлы для отображения в браузере.

ПРИМЕЧАНИЕ

Программисты старой школы и приверженцы HTML должны помнить, что в качестве практически универсального стандарта для корневого каталога документов служит public_html/.

Закройте `show_user.php` и создайте новый сценарий под названием `test.php`. Между открывающим и закрывающим элементами синтаксиса PHP вставьте команду:

```
<?php
echo "Корневой каталог документов: {$_SERVER['DOCUMENT_ROOT']}";
?>
```

$_SERVER — это еще один полезный ассоциативный массив, предоставляемый PHP. Ключ DOCUMENT_ROOT дает возможность узнать о корневом каталоге документов на вашем сервере.

СОВЕТ

Чтобы увидеть, что еще можно получить из массива $_SERVER, посетите сайт www.php.net/manual/ru/reserved.variables.server.php.

Теперь вызовите этот сценарий в браузере. В результате должна появиться картинка, подобная той, что показана на рис. 8.11. В данном примере корневым каталогом является /home1/b/bmclaugh/yellowtagmedia_com. Это значит, что сетевой путь / на самом деле проецируется на путь файловой системы /home1/b/bmclaugh/yellowtagmedia_com.



Рис. 8.11. Корневой каталог документов

Этот проверочный сценарий дает вам нужную отмычку: отображение, связывающее путь файловой системы с фактическим сетевым путем. И показывает, что получить это отображение совсем нетрудно. Для любого путевого имени файла вам нужно убрать все с самого начала, включая yellowtagmedia_com (или включая то, чем заканчивается ваш корневой каталог документов).

Итак, все это необходимо привести в действие. Добавьте путевое имя образца изображения, которое вы только что сохранили в своей базе данных, в сценарий test.php:

```
<?php
echo " Корневой каталог документов: {$_SERVER['DOCUMENT_ROOT']}";
$image_sample_path =
   "/home1/b/bmclaugh/yellowtagmedia_com/phpMM/" .
   "uploads/profile_pics/1312128274-james_roday.jpg";
?>
```

Теперь можно воспользоваться весьма удобной функцией str_replace, которая вам уже хорошо знакома. Вам просто нужно заменить путевое имя, эквивалентное корневому каталогу документов, пустым местом, то есть просто удалить его:

```
<?php
echo "Корневой каталог документов: {$_SERVER['DOCUMENT_ROOT']}";
```

```
$image_sample_path =
   "/home1/b/bmclaugh/yellowtagmedia_com/phpMM/" .
   "uploads/profile_pics/1312128274-james_roday.jpg";
$web_image_path = str_replace($_SERVER['DOCUMENT_ROOT'],
                              '', $image_sample_path);
?>
```

И наконец, нужно вернуть результат с помощью команды echo:

```
<?php
echo "Корневой каталог документов: {$_SERVER['DOCUMENT_ROOT']}";
$image_sample_path =
   "/home1/b/bmclaugh/yellowtagmedia_com/phpMM/" .
   "uploads/profile_pics/1312128274-james_roday.jpg";
$web_image_path = str_replace($_SERVER['DOCUMENT_ROOT'],
                              '', $image_sample_path);

echo "<br /><br />Преобразованный путь: {$web_image_path}";
?>
```

Теперь снова зайдите на свою страницу test.php. Будем надеяться, что вы получите изображение, похожее на то, что показано на рис. 8.12.



Рис. 8.12. Преобразование пути — именно то, что вам нужно: переход от пути в файловой системе, который необходим при работе с этим каталогом изображений, в сетевой путь для вашего браузера

Возьмите этот путь и вставьте его прямо в адресную строку своего браузера, поместив слэш после доменного имени. Затем нажмите клавишу Enter, и, если все пойдет хорошо, вы увидите изображение, над которым так долго работали. Вся эта магия в действии показана на рис. 8.13.

Теперь можно превратить код из test.php в... вы уже догадались: в еще одну полезную сервисную функцию. Откройте свой старый сценарий app_config.php и создайте универсальную версию своего кода из test.php:

```
function get_web_path($file_system_path) {
  return str_replace($_SERVER['DOCUMENT_ROOT'], '', $file_system_path);
}
```

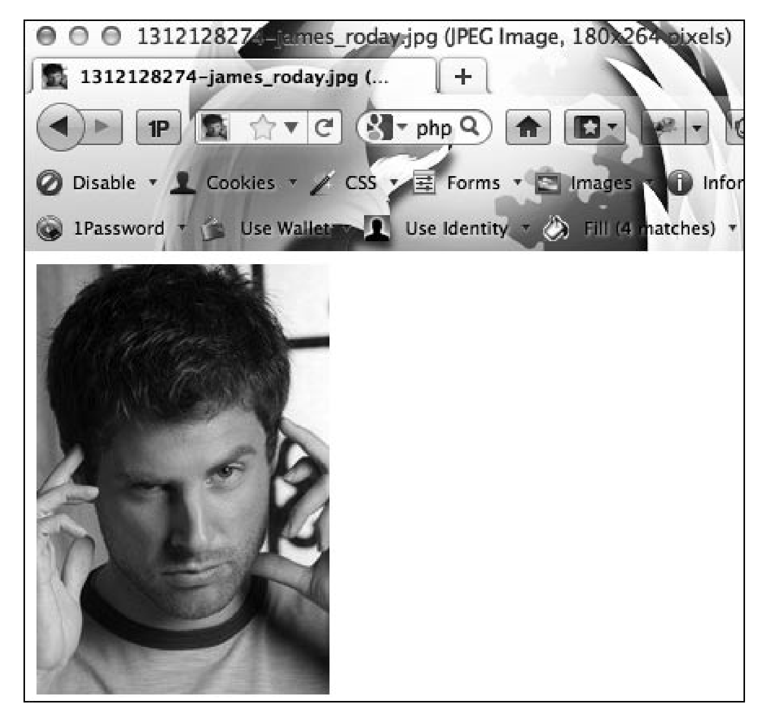


Рис. 8.13. Выкладывание изображения довольно распространенная, но весьма непростая задача, сравнимая по сложности с обработкой ошибок

Очень похоже, не так ли? Вы должны как следует разобраться во всем, что происходит в этом коротком фрагменте кода.

1. С помощью ключевого слова `function` определяется новая функция, которую можно вызвать из любого сценария, затребовав или включив в него `app_config.php`.
2. Функции дается название: `get_web_path`.
3. Определяется единственный элемент информации, которую эта функция получает из вызывающего ее сценария: `$file_system_path`. Это должен быть полный путь от веб-сервера к файлу, и данный путь должен быть преобразован в сетевой.
4. Берется значение переменной `$file_system_path`, и в указанном в нем пути корневой каталог документов заменяется пустым местом (`''`).
5. С помощью инструкции `return` возвращается результат запуска функции `str_replace`.

Новое здесь — лишь инструкция `return`. Она является частью языка PHP и делает именно то, что вы от нее ожидаете: что-то возвращает программе или сценарию, которые вызвали эту функцию. Если функции был передан путь /usr/bbentley/web/images/profile.jpg и корневым каталогом документов был /usr/bbentley/web, то из вызова функции `get_web_path` будет возвращена строка `/images/profile.jpg`.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Создание прототипа в виде простых сценариев

Некоторые языки и среды разработки, такие как, в частности, Ruby on Rails, предлагают средства для запуска команд в контексте вашей среды программирования или сетевой среды. Это некое расширенное окно командной строки, в котором вы получаете все преимущества от запущенного веб-сервера, входа в систему и нескольких других дополнительных свойств среды.

К сожалению, PHP к таким языкам не относится. При тестировании новых функциональных возможностей приходится либо просто прибегать к программированию в одном из уже имеющихся у вас сценариев, либо создавать простой сценарий вроде `test.php` и работать с ним, пока не проявятся эти новые функциональные возможности.

Хотя использование простого сценария, состоящего из одной команды, может показаться чем-то примитивным по сравнению с привлекательной веб-средой со стилем, заданным с помощью CSS, зачастую лучше остановить свой выбор именно на таком сценарии. Вы сможете протестировать нужные возможности и отработать код, не переживая за HTML или за взаимодействие между сценариями. А когда код примет нужные вам кондиции, его можно будет просто вставить в полноценную среду работы сетевых сценариев.

ВНИМАНИЕ

В работе данной функции есть довольно серьезная особенность. Предполагается, что ей передается абсолютный, а не относительный путь. Значит, такой путь, как ../../../web/images/profile.jpg, ни в коем случае не будет соответствовать вашему корневому каталогу документов. К счастью, код, генерирующий путь к изображению, использует абсолютные пути. Следовательно, по крайней мере конкретно для ваших нужд эта функция подойдет как нельзя кстати.

Отображение картинки вашего пользователя: дубль два

Для вывода картинки вашего пользователя нужно вернуться к сценарию `show_user.php`. Но на этот раз у вас уже есть сервисная функция. Примените эту функцию для преобразования абсолютного пути, сохраненного в вашей базе данных, в пригодный для просмотра сетевой путь:

```
if ($result) {
  $row = mysql_fetch_array($result);
  $first_name     = $row['first_name'];
  $last_name      = $row['last_name'];
  $bio            = preg_replace("/[\r\n]+/", "</p><p>", $row['bio']);
  $email          = $row['email'];
  $facebook_url   = $row['facebook_url'];
  $twitter_handle = $row['twitter_handle'];
```

```
  $user_image     = get_web_path($row['user_pic_path']);

  // Превращение $twitter_handle в URL
  $twitter_url = "http://www.twitter.com/" .
                 substr($twitter_handle, $position + 1);

} else {
  handle_error("возникла проблема с поиском вашей " .
               "информации на нашей системе.",
               "Ошибка обнаружения пользователя с ID {$user_id}");
}
```

Проще не бывает. Вернитесь к браузеру и попробуйте либо создать пользователя еще раз (с помощью `create_user.php`), либо посетить страницу `show_user.php` с предоставлением параметра `user_id` в качестве части строки URL. Вы должны увидеть страницу `show_user.php` во всей ее красе: с положенным ей изображением (рис. 8.14).

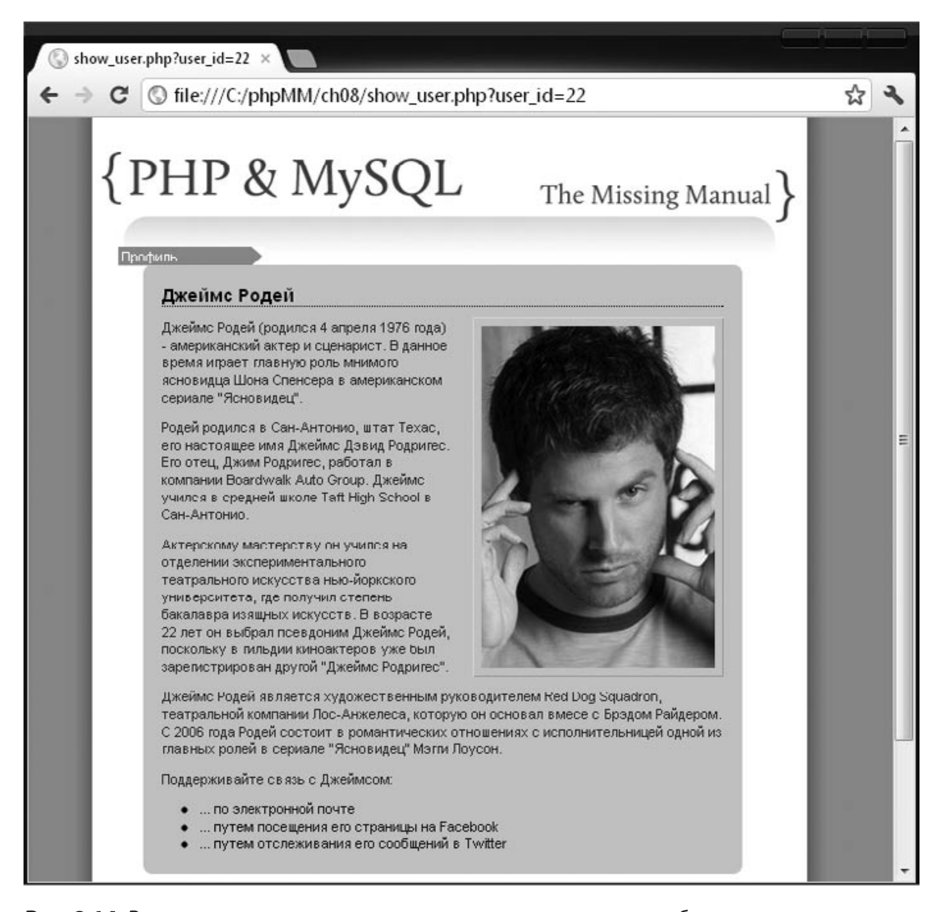


Рис. 8.14. Времени, конечно, ушло немало, но теперь вы можете быть уверены в том, что получили самый безопасный, прекрасно скроенный сценарий, работающий с изображениями

Несколько небольших исправлений в app_config.php

На данный момент у вас есть новый инструмент: $_SERVER['DOCUMENT_ROOT'], а также еще один инструмент на его основе: ваша функция get_web_path. Этой функцией неплохо бы прямо сейчас воспользоваться для того, чтобы привести в порядок сценарий app_config.php. Сейчас корневой каталог вашего сайта определяется примерно следующим образом:

```
// Корневой каталог сайта
define("SITE_ROOT", "/phpMM/");
```

ВНИМАНИЕ

Точное значение для SITE_ROOT будет, наверное, другим. Но у вас все равно будет нечто подобное, что будет похоже на путь от корневого каталога вашего веб-сервера к тому месту, где находятся все файлы сайта. Это может быть просто каталог, обозначаемый символом /, который зачастую и является корневым каталогом сайта.

Но такое определение вносит неоднозначность. Это сетевой путь, а не корневой каталог сайта, который больше похож на конкретный путь в файловой системе. В действительности это путь на машине вашего хостинг-провайдера к корневому каталогу вашего сайта. Поэтому он может иметь вид, похожий на /home1/b/bmclaugh/yellowtagmedia_com. А сетевым путем может быть /, либо /phpMM, или этот путь может иметь вид, зависящий от конкретно вашей структуры каталогов, начинающейся в корневом каталоге.

Но постойте, у вас уже есть кое-что, имеющее вид $_SERVER['DOCUMENT_ROOT']. Но это еще не все, теперь с помощью функции get_web_path вы можете превратить это в сетевой путь, а затем добавить путь от корневого каталога вашего веб-сервера к вашему сайту. Следовательно, если ваша константа SITE_ROOT имела значение /phpMM, то на самом деле вам понадобится что-нибудь еще вроде следующего кода:

```
// Корневой каталог сайта в форме реального пути в файловой системе
define("SITE_ROOT", $_SERVER['DOCUMENT_ROOT'] . "/phpMM/");
```

Это должно превратиться в путь в файловой системе к вашему веб-серверу с последующим добавлением к нему пути от вашего веб-сервера к файлам для данного приложения. В результате получится, к примеру, /home1/b/bmclaugh/yellowtagmedia_com/phpMM. Итак, теперь у вас есть намного более понятная настройка. Вы имеете дело с реальными файлами на реальной файловой системе, а затем, когда вам нужно преобразовать эти файлы и их путевые имена в сетевой домен, вы используете функцию get_web_path.

Чтобы посмотреть, как это работает, проверьте сценарий в функции handle_error. Сейчас у вас должна быть следующая версия:

```
function handle_error($user_error_message, $system_error_message) {
  header("Location: " . SITE_ROOT . "scripts/show_error.php" .
         "?error_message={$user_error_message}" .
         "&system_error_message={$system_error_message}");
}
```

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Почему бы не хранить сетевой путь в базе данных?

В каждом отдельном случае загрузки изображения из базы данных вам приходится вызывать функцию `get_web_path` в отношении пути к изображению, по крайней мере если вам нужно показать изображение, отправив его через Интернет. А поскольку вы занимаетесь созданием веб-приложений, стоит задаться вопросом: как можно упростить создавшуюся ситуацию? Может показаться, что можно просто убрать этот шаг преобразования и просто с самого начала хранить изображение в базе данных в виде сетевого пути.

Но против такой идеи есть несколько доводов. Абсолютный путь не зря так называется. Может измениться программное обеспечение вашего сервера, ваш личный каталог, вы можете перейти с PHP на Ruby, на Perl и обратно на PHP, но, за исключением случая перемещения самого изображения, его абсолютный путь останется неизменным. Еще важнее то, что вы можете вообще изменить корневой каталог документов, а абсолютный путь все равно будет работать.

Почему же этому вопросу придается такое значение? Потому что вам действительно может понадобиться по той или иной причине изменить корневой каталог документов. А если в базе данных будет храниться сетевой путь, то есть путь, зависящий от вашего корневого каталога документов, то при изменении этого каталога все ваши пути к изображениям придут в негодность! Вам придется изменять каждый отдельный путь, перенося его зависимость со старого корневого каталога документов на новый. А это занятие не из приятных.

Кроме того, сетевой путь на самом деле имеет относительный характер, даже если он начинается с символа /. Причина в том, что он связан с вашим корневым каталогом документов. Абсолютный путь установлен относительно машины независимо от ее программного обеспечения. А согласно общему правилу, в базе данных нужно хранить то, что имеет по возможности абсолютный и неизменный характер. Выбирая между элементом информации в абсолютной и в относительной форме, нужно всегда отдавать предпочтение абсолютной форме. Обычно изменить одну форму на другую сравнительно легко, поэтому сохранять нужно наиболее надежную из них.

Но теперь значение константы `SITE_ROOT` уже не является сетевым путем, а вам в данном случае нужен сетевой путь. Вы отправляете перенаправление на браузер, а браузер «разговаривает» на языке сетевых путей, а не файловых систем.

Исправить положение нетрудно, поскольку у вас в `SITE_ROOT` уже имеется реальный путь. Нужно лишь преобразовать его в сетевой путь:

```
function handle_error($user_error_message, $system_error_message) {
  header("Location: " . get_web_path(SITE_ROOT) .
         "scripts/show_error.php" .

         "?error_message={$user_error_message}" .
         "&system_error_message={$system_error_message}");
}
```

Теперь все ваши пути основаны на реально существующих файлах файловой системы. Затем, когда вам придется работать с веб-запросами, нужно будет преобразовать эти реальные пути к файлам в сетевые пути. И, конечно же, это был ключ к получению работоспособных изображений, что стало дополнительным фактором привлекательности приложения. И все же вернемся к этим изображениям.

А теперь совсем о другом

Все заработало, и ваши пользователи могут отправлять изображения на сервер. Вы можете абсолютно безопасно помещать эти изображения в избранное вами постоянное место хранения. Вы получили способ сохранения этого места в базе данных и преобразования записи о нем в URL-адрес, который работает с вашим веб-сайтом и с вашим персональным корневым каталогом документов. А затем, в завершение всего этого, вы получили возможность демонстрации изображений своих пользователей при посещении ими страницы `show_user.php`.

Что же еще можно сделать?

Предположим, что вы применяете несколько веб-серверов, совместно использующих одну и ту же базу данных. Неужели вы станете хранить одно и то же изображение на каждом из этих веб-серверов?

Предположим также, что вы задействуете для веб-сервера какую-нибудь временную машину или допускаете возможность изменений в пользу более высокопроизводительного хостинга по мере расширения своего бизнеса. Хотелось бы вам при этом копировать не только свой сайт, который может занимать по объему всего 10 или 20 Мбайт в сжатом состоянии, но и все изображения ваших пользователей, каждое из которых может составлять 1 или 2 Мбайт? Наверное, нет.

Существуют причины, по которым принятое вами решение может оказаться не самым лучшим для какого-нибудь вашего веб-приложения. Есть еще один вариант, в равной степени и сложный, и полезный: изображения можно хранить не в файловой системе, а непосредственно в базе данных.

Вы располагаете вполне работоспособным решением, но за следующим поворотом может быть еще более удачный вариант. В программировании такое встречается довольно часто. В данном случае речь идет о совершенно другом решении, которое будет рассмотрено в следующей главе. Перелистните страницу и посмотрите, зачем вам может понадобиться хранить в базе данных не путь к изображению, а все изображение целиком.

9 Двоичные объекты и загрузка изображений

На данный момент у вас есть изображения в файловой системе и пути к ним в базе данных. Затем в своем веб-приложении вы превращаете путь в этой файловой системе в сетевой путь и показываете изображение. И все это довольно неплохо работает. Можно, конечно, пользоваться этим решением и никогда не столкнуться ни с какими проблемами, но, возможно, пройдет несколько недель и вы столкнетесь с очень большой проблемой.

Недостатком такого подхода является отсутствие самодостаточного решения. Изображения находятся в файловой системе, пути располагаются в базе данных, а затем вам нужен код PHP для превращения местонахождения изображения на вашем веб-сервере в путь, который может быть правильно интерпретирован браузерами ваших пользователей. То есть вы создали настоящую так называемую тесную связь между файловой системой, PHP и базой данных.

Итак, как же сделать все это более самодостаточным? Вам придется взять все эти элементы информации и поместить их в одно место. Понятно, что без базы данных тут не обойтись, поэтому она становится тем самым логическим местом, где будет консолидирована вся ваша информация. Сюда будет попадать не просто часть, а вся информация.

При таком подходе в базу данных помещается не просто ссылка на отправленное на сервер изображение, а само это изображение. Но для этого нужно проделать большой объем дополнительной работы: вам нужен не просто новый столбец в таблице `users`, а совершенно новая таблица. Вам необходим новый тип данных, и вам понадобятся не только `SELECT`- и `INSERT`-запросы, использовавшиеся до сих пор. По большому счету вам нужна только модель решения, поэтому давайте и займемся этим вопросом.

Хранение разных объектов в различных таблицах

До сих пор вы работали только с одной таблицей — `users`. Это объясняется простотой задачи: представлением одного из ваших пользователей. Все в данной таблице — имя и фамилия, адрес электронной почты, URL-адрес Facebook и идентификатор в Twitter — являются частями сведений об этом пользователе. Можно также сказать, что все в таблице `users` является описанием пользователя.

А теперь вы пришли к идее сохранения изображения и информации *об этом* изображении, то есть того, что представляется вам важным, когда вы собираетесь хранить все это изображение в базе данных. Этот контент уже не относится к описанию пользователя.

Фактически изображение, которое пользователь хочет показать при просмотре своего профиля и которое связано с этим пользователем, становится самостоятельным объектом.

И поскольку вы дошли до этого момента осознания работы с новым объектом, вам нужно создать новую таблицу по имени `images`, в которой будут храниться не только изображение пользователя, но и ключевые детали, касающиеся этого изображения.

ПРИМЕЧАНИЕ

Если подумать, данная таблица похожа на таблицу users. В той таблице хранится не только имя пользователя (которое можно принять за эквивалент самого изображения), но и информация об этом пользователе, такая как адрес его электронной почты и идентификатор в Twitter. Аналогично этому вы собираетесь хранить информацию об изображении, помогающую определить, когда его нужно использовать.

- **ID изображения.** Идентифицирует данное изображение, что очень похоже на `user_id` в таблице `users`. Чуть позже это поле также позволит связать изображение с таблицей `users`.
- **Имя изображения.** Несмотря на то что в таблице будут храниться данные изображения, вам все равно нужно будет сохранить его имя, по которому на него можно будет ссылаться.
- **MIME-тип.** Эта информация важна для сообщения веб-серверу о типе выводимых на экран данных: JPG, GIF, PNG и т. д.
- **Размер файла.** Используется браузером для отображения изображения.
- **Данные изображения.** Простые биты и байты, которые превращаются в пикселы и цвета.

Переведите все это в SQL и получите новую инструкцию `CREATE`:

```
CREATE TABLE images (
  image_id    int           AUTO_INCREMENT PRIMARY KEY,
  filename    varchar(200) NOT NULL,
  mime_type   varchar(50)  NOT NULL,
  file_size   int          NOT NULL,
  image_data  mediumblob   NOT NULL
);
```

Все эти элементы не вызывают вопросов за исключением нового типа столбца: `mediumblob`. Вообще-то существуют и другие блоб-типы[1]:

- `tinyblob` — для хранения объектов до 256 байт;
- `blob` — для хранения объектов до 65 Кбайт в `blob`-столбце;

---

[1] Блоб (Blob, Binary Large Object) означает большой двоичный объект. Это столбец, созданный именно для того типа информации, из которого состоит изображение: информации, которая относится не к числу или строке, а к двоичным данным.

- mediumblob — для хранения до 16 Мбайт данных;
- longblob — самый объемный блоб-тип для хранения 4 Гбайт данных в longblob-столбце.

ЗАМЕТКИ О ПРОЕКТИРОВАНИИ

Планирование роста ваших данных и их описание

Существуют серьезные разногласия по поводу того, какой из блоб-типов нужно использовать. Одни утверждают, что практически всегда необходимо применять longblob. В то время как другие говорят, что следует выяснить всю информацию о данных и воспользоваться тем размером блоба, который вмещает их размер, только и всего.

Для приверженцев постоянного использования longblob аргументом служит опережающее планирование. Поскольку база данных использует пространство, необходимое для фактической информации, а не максимальный размер столбца — столбец типа longblob, который содержит изображение объемом 2 Мбайт, будет занимать ровно столько же места, сколько и столбец типа mediumblob, включающий в себя изображение в 2 Мбайт. Если использовать longblob повсеместно, то при необходимости изменения объема хранилища не придется менять тип столбца.

С другой стороны, если разрешаются только изображения размером не более 2 Мбайт, то лучше дать описание данных путем использования mediumblob. Тогда вы не только выберете произвольный тип, но и предоставите информацию о том, что попадает в столбец.

Если будет сохраняться только имя, вряд ли нужно выбирать для всех столбцов тип varchar(255), поскольку таких длинных имен не существует. Вы при таком подходе упустите шанс сообщить что-нибудь о своих данных. Это же справедливо для использования longblob, если (и это весьма важное «если») вы решили, что будете принимать только те изображения, которые по размеру будут помещаться в столбец с типом данных mediumblob.

Итак, создайте данную таблицу в той же базе данных, в которой находится таблица users. Теперь вы можете увидеть в своей базе данных обе эти таблицы:

```
mysql> USE bmclaugh;
Database changed
mysql> SHOW tables;
+-----------------------------------+
| Tables_in_bmclaugh                |
+-----------------------------------+
| images                            |
| users                             |
+-----------------------------------+
2 rows in set (0.00 sec)
```

Вставка в таблицу необработанного изображения

Настало время еще раз посетить create_user.php. Будет использована основная часть этого сценария, но при этом нужно внести некоторые изменения. В частности, подойдут все проверки, которые гарантируют, что пользователь отправил настоящее изображение, что сервер или PHP не сгенерировали ошибки и что файл является изображением (это определяется с помощью функции getimagesize).

Изменения касаются того фрагмента кода, который используется для перемещения временного изображения в постоянное место хранения. При данном подходе постоянным местом является таблица images, следовательно, этот код нужно заменить.

Вот как выглядит сценарий create_user.php с кодом, больше не нуждающимся в удалениях.

ПРИМЕЧАНИЕ

Перед внесением изменений нужно сделать резервную копию create_user.php. Скопируйте его в файл create_user.php.bak или в какой-нибудь похожий файл, чтобы при желании можно было легко вернуться назад к хранению пути к изображению.

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';

// Массив ошибок и переменные, связанные с изображениями, остаются прежними
$upload_dir = SITE_ROOT . "uploads/profile_pics/";
$image_fieldname = "user_pic";

// Потенциальные PHP-ошибки отправки файлов
$php_errors = array(1 => 'Превышен макс. размер файла, указанный в php.ini',
                2 => 'Превышен макс. размер файла, указанный в форме HTML',
                3 => 'Была отправлена только часть файла',
                4 => 'Файл для отправки не был выбран.');

$first_name = trim($_REQUEST['first_name']);
$last_name = trim($_REQUEST['last_name']);
$email = trim($_REQUEST['email']);
$bio = trim($_REQUEST['bio']);
$facebook_url = str_replace("facebook.org", "facebook.com", trim($_
REQUEST['facebook_url']));
$position = strpos($facebook_url, "facebook.com");
if ($position === false) {
  $facebook_url = "http://www.facebook.com/" . $facebook_url;
}

$twitter_handle = trim($_REQUEST['twitter_handle']);
$twitter_url = "http://www.twitter.com/";
$position = strpos($twitter_handle, "@");
```

```
if ($position === false) {
  $twitter_url = $twitter_url . $twitter_handle;
} else {
  $twitter_url = $twitter_url .
                 substr($twitter_handle, $position + 1);
}

// Проверка отсутствия ошибки при отправке изображения
($_FILES[$image_fieldname]['error'] == 0)
  or handle_error("сервер не может получить выбранное вами изображение.",
                  $php_errors[$_FILES[$image_fieldname]['error']]);

// Является ли этот файл результатом нормальной отправки?
@is_uploaded_file($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы попытались совершить безнравственный поступок. Позор!",
                  "Запрос на отправку: файл назывался " .
                  "'{$_FILES[$image_fieldname]['tmp_name']}'");

// Действительно ли это изображение?
@getimagesize($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы выбрали файл для своего фото, " .
                  "который не является изображением.",
                  "{$_FILES[$image_fieldname]['tmp_name']} " .
                  "не является настоящим файлом изображения.");

// Присваивание файлу уникального имени
$now = time();
while (file_exists($upload_filename = $upload_dir . $now .
                                     '-' .
                   $_FILES[$image_fieldname]['name'])) {
    $now++;
}

// Удалите код, использующий move_uploaded_file для перемещения
// временного изображения

// Удалите имя и значение столбца для пользовательских изображений
$insert_sql = "INSERT INTO users (first_name, last_name, email, " .
              "bio, facebook_url, twitter_handle) " .
              "VALUES ('{$first_name}', '{$last_name}', '{$email}', " .
              "'{$bio}', '{$facebook_url}', '{$twitter_handle}');";

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . mysql_insert_id());
exit();
?>
```

Итак, код остался практически таким же. Основное изменение заключается в том, что теперь вам нужна новая инструкция INSERT, которая будет вставлять данные не в таблицу users, а в таблицу images.

И здесь проявляется вся прелесть этого решения: вы можете получить каждую необходимую порцию информации, которую нужно поместить в таблицу images из массива $_FILES (который на самом деле является массивом массивов). С этих позиций создание кода не представляет особых трудностей:

```
$insert_sql = "INSERT INTO users (first_name, last_name, email, " .

              "bio, facebook_url, twitter_handle) " .
              "VALUES ('{$first_name}', '{$last_name}', '{$email}', " .
              "'{$bio}', '{$facebook_url}', '{$twitter_handle}');";

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Вставка изображения в таблицу images
$image = $_FILES[$image_fieldname];
$image_filename = $image['name'];
$image_info = getimagesize($image['tmp_name']);
$image_mime_type = $image_info['mime'];
$image_size = $image['size'];
$image_data = file_get_contents($image['tmp_name']);

$insert_image_sql = "INSERT INTO images " .
                            "(filename, mime_type, file_size, image_data) " .
                    "VALUES ('{$image_filename}', '{$image_mime_type}', " .
                            "'{$image_size}', '{$image_data}');";

mysql_query($insert_image_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . mysql_insert_id());
?>
```

Этот код богат на действия, в которых можно запросто запутаться, поэтому разберем все по частям.

Сначала просто для удобства создается новая переменная $image:

```
$image = $_FILES[$image_fieldname];
```

Эта строка упрощает работу со всеми свойствами изображения. Вам не нужно снова и снова набирать $_FILES[$image_fieldname]. Этот шаг не носит обязательного характера, но существенно все упрощает.

Затем из данного массива можно получить имя изображения:

```
$image_filename = $image['name'];
```

Функция getimagesize не возвращает размер файла

И вот здесь начинаются странности. Функция getimagesize вообще-то возвращает не числовое значение размера файла отправленного изображения, а массив информации об этом изображении, содержащий MIME-тип (который вам нужен), а также высоту и ширину изображения, которые можно использовать для вывода изображения в HTML-странице (в которых пока нет надобности).

Вы можете подумать, что нужно сделать нечто подобное:

```
$image_size = getimagesize($image['tmp_name']);
```

Но при этом столкнетесь с двумя проблемами: getimagesize не размер, а массив, а размеры, возвращаемые этой функцией, являются высотой и шириной, а не размером файла.

А от возвращенного массива вам нужен MIME-тип:

```
$image_info = getimagesize($image['tmp_name']);
$image_mime_type = $image_info['mime'];
```

Но вам по-прежнему нужен размер файла отправленного изображения. Его следует получить из свойства исходного массива, связанного с изображением:

```
$image_size = $image['size'];
```

Функция file_get_contents оправдывает свое название

Иногда название функции вводит в заблуждение, как это было с функцией getimagesize. Но бывает и так, что функция имеет весьма удачное название, что в полной мере можно отнести к функции file_get_contents. Эта функция предоставляет вам данные из объекта в двоичной форме, что, собственно, и нужно для столбца image_data в таблице images:

```
$image_data = file_get_contents($image['tmp_name']);
```

Вставка изображения с помощью инструкции INSERT

И последнее, что нужно сделать по очереди, но не по значению — создать запрос на вставку INSERT и запустить его:

```
$insert_image_sql = "INSERT INTO images " .
                        "(filename, mime_type, file_size, image_data) " .
                   "VALUES ('{$image_filename}', '{$image_mime_type}', " .
                        "'{$image_size}', '{$image_data}');";

mysql_query($insert_image_sql);
```

ВНИМАНИЕ

Воздержитесь от запуска этого кода! Если же он будет запущен, приготовьтесь получить ряд весьма странных ошибок. Возникающие при этом проблемы относятся к темным сторонам обработки данных со стороны MySQL. Поэтому приведите код в данное состояние и продолжите чтение, пока у вас не появилась мысль о каких-нибудь ошибочных действиях.

Пока ваши двоичные данные вставлять небезопасно

Похоже, дела идут неплохо, но если вы запустите код из предыдущего раздела, возникнут проблемы. Прежде всего двоичные данные содержат всевозможные странные символы, на которых могут застопориться PHP и MySQL. Вероятность столкнуться с проблемными символами имеется практически всегда, и это особенно актуально и почти не имеет исключений при работе с двоичными данными.

И тут опять нам на помощь приходит очередная полезная функция.

ПРИМЕЧАНИЕ

Вы, наверное, уже заметили, что на каждом повороте нас поджидает какая-нибудь полезная функция PHP. Это одно из весьма существенных преимуществ данного языка. Для версий 4 и 5 языка PHP была разработана и установлена весьма надежная библиотека полезных функций, к которым относится как функция getimagesize, так и функция mysql_real_escape_string, которой вы сейчас воспользуетесь.

Функция mysql_real_escape_string нейтрализует любые специальные символы в той строке, которая ей передается. Следовательно, вы можете передать свою переменную $image_data, а затем результат выполнения функции mysql_real_escape_string передать функции mysql_query через вашу инструкцию INSERT. Неплохо будет применить функцию к любым строковым данным, передаваемым MySQL:

```
$insert_sql = "INSERT INTO users (first_name, last_name, email, " .
              "bio, facebook_url, twitter_handle) " .
              "VALUES ('{mysql_real_escape_string($first_name)}', " .
                      "'{mysql_real_escape_string($last_name)}', " .
                      "'{mysql_real_escape_string($email)}', " .
                      "'{mysql_real_escape_string($bio)}', " .
                      "'{mysql_real_escape_string($facebook_url)}', " .
                      "'{mysql_real_escape_string($twitter_handle)}');";

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Вставка изображения в таблицу images
$image = $_FILES[$image_fieldname];
$image_filename = $image['name'];
$image_info = getimagesize($image['tmp_name']);
$image_mime_type = $image_info['mime'];
$image_size = $image['size'];
```

```
$image_data = file_get_contents($image['tmp_name']);

$insert_image_sql = "INSERT INTO images " .
                         "(filename, mime_type, file_size, image_data) " .
                   "VALUES ('{mysql_real_escape_string($image_filename)}', ".
                         "'{mysql_real_escape_string($image_mime_type)}', " .
                         "'{ mysql_real_escape_string($image_size)}', " .
                         "'{mysql_real_escape_string($image_data)}');";

mysql_query($insert_image_sql);
```

ПРИМЕЧАНИЕ

Поскольку переменная $image_size имеет числовое значение, ее не нужно передавать функции mysql_real_escape_string. Но если вы постоянно пытаетесь запомнить, являются или нет введенные данные строкой или числом, то в конечном итоге это может привести к ошибке и к отсутствию нужной нейтрализации специальных символов.

В целях безопасности лучше нейтрализовать специальные символы в значениях всех переменных. Это более последовательный подход, являющийся еще одним уровнем защиты. Время, затрачиваемое PHP на нейтрализацию этого элемента данных, не играет особой роли по сравнению с теми проблемами, которые возникнут, если не будут нейтрализованы злонамеренно внедренные специальные символы.

Вставка строки в переменную

При всей естественности данного кода в нем кроется серьезная проблема. Поскольку фигурные скобки, окружающие переменную, позволяют значению этой переменной быть вставленным в строку (то есть выражение `"{$variable}"` вставляет значение переменной `$variable`), PHP накладывает ограничения на выполнение каких-нибудь действий внутри фигурных скобок. Поэтому код не будет интерпретироваться как вызов `mysql_real_escape_string`.

Обойти эту дилемму можно двумя способами. Проще всего воспользоваться первым из них: просто переместить вызовы функции `mysql_real_escape_string` в присваивания значений переменным:

```
// Вставка изображения в таблицу images
$image = $_FILES[$image_fieldname];
$image_filename = mysql_real_escape_string($image['name']);
$image_info = getimagesize($image['tmp_name']);
$image_mime_type = mysql_real_escape_string($image_info['mime']);
// и т. д...
```

Выглядит хорошо, но сама идея не из лучших. Подумайте о вызываемой функции: она конкретно предназначена для настройки переданных значений на работу с MySQL. Что если значение переменной `$image_filename` нужно будет использовать в каком-нибудь другом месте вашего сценария, а вы уже превратили его в версию имени файла, специально предназначенную для MySQL?

Похоже, правильным будет первоначальный подход — конвертация значения с использованием функции `mysql_real_escape_string` непосредственно перед пере-

дачей его SQL-инструкции INSERT. Этот подход позволяет переменной быть просто именем файла изображения или MIME-типом изображения, а затем, когда это потребуется, конвертировать ее значение в форму, предназначенную для работы с MySQL.

Похоже, все указывает на то, что нужен способ для вычислений или для запуска функций в отношении значений, используемых при создании строки SQL, и он у нас есть. Практически то же самое делается при использовании PHP-функции sprintf, которая выводит строку. Иными словами, строка составляется с применением любых нужных вычислений и с передачей всей требуемой информации функции sprintf. Эта функция собирает все вместе и возвращает строку, которую затем можно присвоить вашей переменной, после чего она будет готова для передачи функции mysql_query.

Эта технология немного отличается от всего, что вы делали до сих пор. Вместо составления строки с помощью объединения показывается вся создаваемая строка, но в тех местах, куда нужно включить значение переменной, помещаются специальные спецификаторы типа. Например, **для строкового типа используется спецификатор** %s:

```
$hello = sprintf("Hello there, %s %s", $first_name, $last_name);
echo $hello;
```

Предположим, именем ($first_name) является John а фамилией ($last_name) — Wayne. В результате запуска этого сценария из двух строк будет получена следующая строка:

```
Hello there, John Wayne
```

Функция sprintf заменяет первый спецификатор %s первым значением, указанным после строки, которым является значение переменной $first_name. Затем она заменяет второй спецификатор %s вторым значением, приведенным после строки, то есть значением переменной $last_name. И наконец, вся строка со вставками присваивается переменной $hello.

Преимущество от использования sprintf заключается в возможности выполнения вычислений над переменными перед передачей их значений функции sprintf. И следующий код при всей его примитивности абсолютно легален:

```
$hello = sprintf("Hello there, %s", $first_name . ' ' . $last_name);
echo $hello;
```

Разумеется, есть куда более оптимальные способы использования sprintf, например создание строки запроса с применением функции mysql_real_escape_string:

```
// Этот код заменяет прежнее присваивание значения переменной $insert_sql
$insert_sql = sprintf("INSERT INTO users " .
                       "(first_name, last_name, email, " .
                        "bio, facebook_url, twitter_handle) " .
                  "VALUES ('%s', '%s', '%s', '%s', '%s', '%s');",
                  mysql_real_escape_string($first_name),
                  mysql_real_escape_string($last_name),
```

```
                mysql_real_escape_string($email),
                mysql_real_escape_string($bio),
                mysql_real_escape_string($facebook_url),
                mysql_real_escape_string($twitter_handle));

// Вставка пользователя в базу данных
mysql_query($insert_sql);
```

Результат работы этого кода ничем существенным не отличается от результата работы старой версии, потому что данные, вставляемые в запись пользователя, не являлись первоначально решаемой проблемой. Но теперь вы можете применить тот же самый подход и к вставке записи в таблицу images:

```
$insert_image_sql = sprintf("INSERT INTO images " .
                            "(filename, mime_type, " .
                             "file_size, image_data) " .
                    "VALUES ('%s', '%s', %d, '%s');",
                    mysql_real_escape_string($image_filename),
                    mysql_real_escape_string($image_mime_type),
                    mysql_real_escape_string($image_size),
                    mysql_real_escape_string($image_data));

mysql_query($insert_image_sql);
```

Можно догадаться, что именно спецификатор %d означает для функции sprintf: он означает, что этот спецификатор типа нужно заменить десятичным числом, таким как 1024 или 92 048. Таким образом, этот код составляет инструкцию INSERT и выполняет ее, нейтрализуя в процессе этого все специальные символы.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Теперь sprintf — ваш новый лучший друг

Большинство PHP-программистов используют функцию sprintf главным образом потому, что она позволяет им предпринимать такие действия, как использование mysql_real_escape_string в отношении переменных перед их вставкой в строку запроса. Но те же самые программисты приходят к таким же выводам, к которым можете прийти и вы: применение sprintf позволяет создавать более надежный и гибкий код.

Теперь вы можете проводить вычисления над своими данными, нейтрализовать в значениях специальные символы и совершать с вашими данными множество других необходимых вам действий при их вставке в базу данных или извлечении из нее. Теперь больше не придется производить вычисление, потом присваивать результат этих вычислений переменной (или, что еще хуже, новой переменной, значение которой основано на значении какой-нибудь старой переменной), а затем, и только тогда, использовать такие переменные, как часть SQL-конструкции.

Функция sprintf позволяет делать все за один шаг. Вообще-то ее нужно использовать как исходное средство создания SQL-строк, выполняемых в качестве запросов к базе данных.

Теперь испытайте этот сценарий в работе. Перейдите еще раз к create_user.php, попросите своего коллегу заполнить форму, выбрать изображение и отправить данные на сервер. При этом должна запуститься ваша новая версия create_user.php и вы должны быть перенаправлены на show_user.php.

На этот раз вы не увидите новый профиль пользователя, поскольку вы еще не написали соответствующий код. Но вы можете обратиться к новой таблице images и увидеть запись для отправленного на сервер изображения:

```
mysql> SELECT image_id, filename FROM images;
+----------+---------------------------+
| image_id | filename                  |
+----------+---------------------------+
|        4 | 220px-William_Shatner.jpeg |
+----------+---------------------------+
1 row in set (0.00 sec)
```

В данном случае вам вряд ли захочется воспользоваться инструкцией SELECT *, поскольку вы столкнетесь с попыткой MySQL загрузить данные вашего изображения, которые могут иметь длину в несколько сотен или тысяч килобайт! Но по крайней мере вы увидите, что изображение действительно находится в вашей таблице.

Можно также обратиться к таблице с помощью PhpMyAdmin, если это средство у вас запущено, и получить дополнительную информацию о ваших записях в таблице images (рис. 9.1). PhpMyAdmin выводит отчет о блоб-столбцах независимо от используемого блоб-типа в виде слова BLOB и указания размера. В данном случае можно увидеть, что размер файла составляет 11 729 байт, что соответствует размеру данных в блоб-столбце, составляющему 11,5 Кбайт. Это хороший способ убедиться в работоспособности сценария: он исправно получает размер изображения, вставляемого в вашу базу данных.

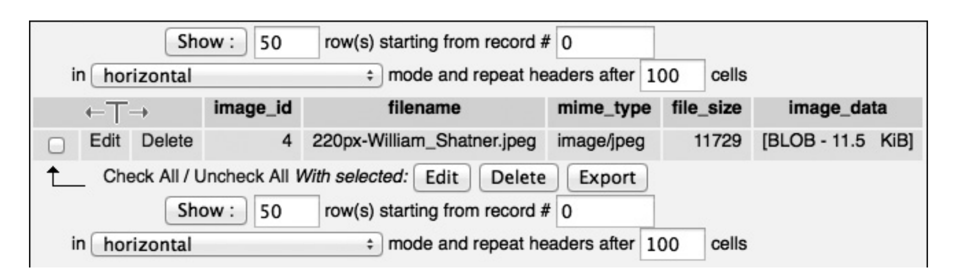


Рис. 9.1. Отчет PhpMyAdmin

Получение правильного ID перед перенаправлением

К сожалению, у нас есть нерешенная проблема. При вставке изображения вы можете получить примерно такую же картину, которая показана на рис. 9.2.

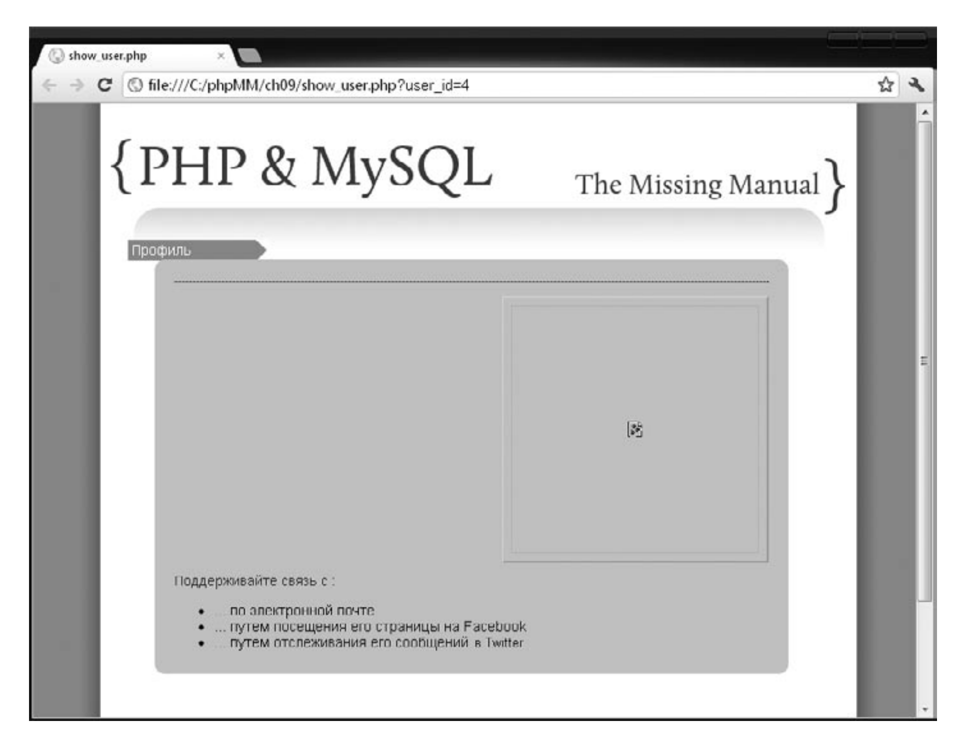


Рис. 9.2. Возникла какая-то проблема

Этот экран вряд ли соответствует тому, что вы хотели увидеть после всей той работы, которая была проделана для помещения изображений в вашу базу данных. Что же случилось?

Пропавшее изображение не является каким-то таинственным событием, как это может показаться на первый взгляд. Вот последний фрагмент кода из `create_user.php`:

```
// Этот код заменяет прежнее присваивание значения переменной $insert_sql
$insert_sql = sprintf("INSERT INTO users " .
                       "(first_name, last_name, email, " .
                        "bio, facebook_url, twitter_handle) " .
                  "VALUES ('%s', '%s', '%s', '%s', '%s', '%s');",
                  mysql_real_escape_string($first_name),
                  mysql_real_escape_string($last_name),
                  mysql_real_escape_string($email),
                  mysql_real_escape_string($bio),
                  mysql_real_escape_string($facebook_url),
                  mysql_real_escape_string($twitter_handle));

// Вставка пользователя в базу данных
mysql_query($insert_sql);

$insert_image_sql = sprintf("INSERT INTO images " .
                             "(filename, mime_type, " .
```

```
                              "file_size, image_data) " .
                         "VALUES ('%s', '%s', %d, '%s');",
                         mysql_real_escape_string($image_filename),
                         mysql_real_escape_string($image_mime_type),
                         mysql_real_escape_string($image_size),
                         mysql_real_escape_string($image_data));

mysql_query($insert_image_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . mysql_insert_id());
```

В чем проблема? В последней строке кода. Вспомните, mysql_insert_id возвращает ID последнего запроса INSERT, который больше уже не является запросом INSERT для вашей таблицы users, а относится к новому запросу INSERT к таблице images. Следовательно, перенаправление на show_user.php работает, но при этом отправляется ID вставленного изображения, а не пользователя. Это можно легко исправить:

```
// Этот код заменяет прежнее присваивание значения переменной $insert_sql
$insert_sql = sprintf("INSERT INTO users " .
                          "(first_name, last_name, email, " .
                           "bio, facebook_url, twitter_handle) " .
                   "VALUES ('%s', '%s', '%s', '%s', '%s', '%s');",
                   mysql_real_escape_string($first_name),
                   mysql_real_escape_string($last_name),
                   mysql_real_escape_string($email),
                   mysql_real_escape_string($bio),
                   mysql_real_escape_string($facebook_url),
                   mysql_real_escape_string($twitter_handle));

// Вставка пользователя в базу данных
mysql_query($insert_sql);

$user_id = mysql_insert_id();

$insert_image_sql = sprintf("INSERT INTO images " .
                             "(filename, mime_type, " .
                              "file_size, image_data) " .
                         "VALUES ('%s', '%s', %d, '%s');",
                         mysql_real_escape_string($image_filename),
                         mysql_real_escape_string($image_mime_type),
                         mysql_real_escape_string($image_size),
                         mysql_real_escape_string($image_data));

mysql_query($insert_image_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . $user_id);
exit();
```

Еще раз попробуйте запустить этот сценарий, и вы вернетесь к ожидаемому результату: к немного подпорченной версии show_user.php, но этот дефект вполне ожидаем (рис. 9.3).

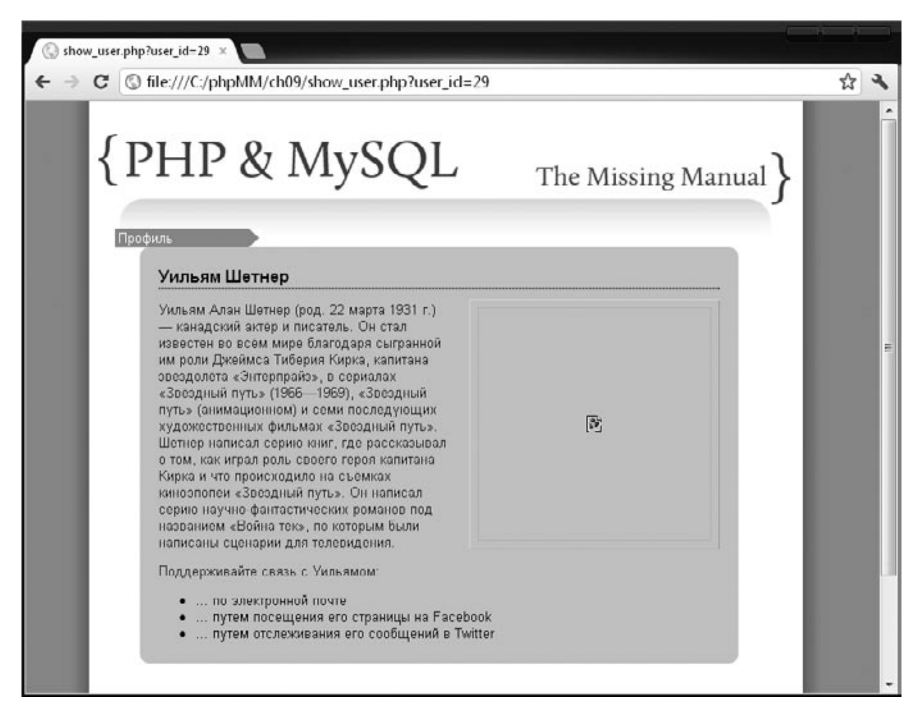


Рис. 9.3. Теперь информация о пользователе есть

Как бы странно это ни звучало, но иногда нужно, чтобы кое-что было подпорчено. В данном случае вы хотели увидеть отсутствие изображения, поскольку вы не написали код для показа того изображения, которое только что было вставлено с помощью инструкции INSERT. А вот что вы не хотели увидеть и что только что было исправлено, так это отсутствие информации о пользователе, за исключением изображения.

Связывание пользователей и изображений

Здесь мы столкнулись с трудностью. У вас есть две таблицы — users и images, но нет связи между ними. Как при загрузке данных пользователя и отображения их путем использования show_user.php определить, какое изображение из таблицы images нужно вывести для заданного пользователя из таблицы users?

Понятно, что необходима какая-то связь между этими двумя таблицами. У вас уже есть уникальный ID для каждой записи в таблице users (user_id) и в таблице images (image_id), что является хорошей стартовой позицией. Но связан ли пользователь с изображением или изображение с пользователем?

Фундаментальный вопрос, задаваемый всякий раз при связывании двух таблиц в базе данных, звучит так: чем связаны эти две таблицы? Или, если точнее, как связаны друг с другом два объекта, представляемые этими таблицами?

Есть ли у пользователя изображение? Есть ли у пользователя несколько изображений? В данном случае у одного пользователя есть одно изображение профиля. То есть у вас есть то, что называется связью «один к одному» (или 1–1). Один пользователь связан с одним изображением. Следовательно, вы можете создать новый столбец в своей таблице users, и в этом столбце вы можете хранить image_id изображения профиля этого пользователя. Вы можете внести это изменение в свою базу данных:

```
mysql> ALTER TABLE users
    ->         ADD profile_pic_id int;
Query OK, 6 rows affected (0.11 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

ВНИМАНИЕ

Вы уже внесли изменения в свои сценарии для обеспечения хранения изображений в своей базе данных, а не в своей файловой системе. Теперь с помощью инструкции ALTER вы вносите аналогичные изменения в свою базу данных. Эти изменения отражают изменения в работе вашего приложения. Поэтому в данный момент вам нужно создать резервную копию своей базы данных.

Но создать резервную копию сценария намного проще, чем сделать то же самое с базой данных. Возможно, вам придется позвонить в компанию, предоставляющую хостинг, и узнать, можно ли создать резервную копию базы данных и как это сделать. Или же вы должны выяснить, как можно отменить эти изменения, если вы вернетесь к хранению изображений в своей файловой системе.

В любом случае вы должны получить некоторые практические навыки в работе с PHP и MySQL по переключению между этими двумя подходами. Это пригодится вне зависимости от того, на использовании какого именно решения вы в конечном счете остановитесь.

ЗАМЕТКИ О ПРОЕКТИРОВАНИИ

Внешние ключи и имена столбцов

Столбец profile_pic_id настроен на связь, называемую внешним ключом. Этот столбец является внешним ключом, поскольку он связан с ключом во «внешней» таблице images.

В большинстве баз данных вы не только определяете столбец в своей таблице, который связан с первичным ключом таблицы, на которую делается ссылка, но и определяете внешний ключ — FOREIGN KEY на уровне базы данных. Теперь ваша база данных будет знать, что в столбце profile_pic_id хранятся ID, которые находятся в столбце image_id таблицы images. Но до недавних времен MySQL не предоставляла такое свойство.

Теперь в MySQL можно использовать внешний ключ, но вам придется применять имеющийся в MySQL механизм работы с таблицами InnoDB. Для этого понадобятся дополнительные настройки, и не все хостинг-провайдеры поддерживают InnoDB. Более того, программисты годами использовали MySQL без поддержки внешнего ключа, поэтому при соответствующем программировании можно обойти это

ограничение. Если вы хотите задействовать InnoDB и поддержку внешнего ключа на уровне базы данных, то при работе с таблицами начните со следующей команды:

```
ALTER TABLE [table-name]
      ENGINE = InnoDB;
```

Затем запустите поиск Google по ключевой фразе MySQL foreign keys, и вы получите в свое распоряжение огромный объем информации.

Независимо от того, применяете вы в своем программировании внешние ключи или нет или же добавляете поддержку на уровне базы данных путем использования InnoDB, выбор имени столбца внешнего ключа имеет важное значение. В соответствии со сложившейся практикой столбцу внешнего ключа присваивается имя вида [имя-таблицы-в-единственном-числе]_id. Следовательно, для внешнего ключа, связывающего таблицу users с таблицей images, обычно берется в единственном числе имя той таблицы, с которой устанавливается связь — image от images — и добавляется _id. В результате именем столбца внешнего ключа становится image_id.

Тогда почему же в таблице users используется имя profile_pic_id? Поскольку вы с таким же успехом можете в таблице images хранить не только картинки профилей. Вы можете хранить для пользователя несколько изображений, и только одно из них будет картинкой профиля. Там могут храниться личные фотографии, или значки для входа, или изображения, относящиеся к компаниям, с которыми связаны ваши пользователи.

Тогда во всех этих случаях столбец image_id в таблице users не предоставит достаточной специфичности. В подобных случаях, когда не просто устанавливается внешний ключ, а, кроме того, еще и указывается конкретный тип использования, есть смысл применить другое имя. Например, вы можете остановиться на имени столбца profile_pic_id в таблице users, а затем, возможно, на имени столбца company_logo_id в возможной таблице companies, и кто знает, какие еще изображения вы будуете применять? Используя в данном случае имя profile_pic_id, вы показываете, что оно связано с изображением (в нем есть сокращение pic - изображение) и что это изображение предназначено для профиля пользователя (в нем есть слово profile — профиль).

Вставка изображения при вставке пользователя

Подумайте как следует о том, что у вас здесь имеется. Поскольку изображение находится в таблице images, вам нужно получить ID этого изображения и вставить его в столбец profile_pic_id таблицы users. Но в данный момент ваш сценарий осуществляет вставку в таблицу users перед вставкой в таблицу images:

```
// Этот код заменяет прежнее присваивание значения переменной $insert_sql
$insert_sql = sprintf("INSERT INTO users " .
                        "(first_name, last_name, email, " .
                         "bio, facebook_url, twitter_handle) " .
                 "VALUES ('%s', '%s', '%s', '%s', '%s', '%s');",
                 mysql_real_escape_string($first_name),
                 mysql_real_escape_string($last_name),
```

```
                        mysql_real_escape_string($email),
                        mysql_real_escape_string($bio),
                        mysql_real_escape_string($facebook_url),
                        mysql_real_escape_string($twitter_handle));

// Вставка пользователя в базу данных
mysql_query($insert_sql);

$user_id = mysql_insert_id();

$insert_image_sql = sprintf("INSERT INTO images " .
                                         "(filename, mime_type, " .
                                          "file_size, image_data) " .
                               "VALUES ('%s', '%s', %d, '%s');",
                               mysql_real_escape_string($image_filename),
                               mysql_real_escape_string($image_mime_type),
                               mysql_real_escape_string($image_size),
                               mysql_real_escape_string($image_data));

mysql_query($insert_image_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . $user_id);
exit();
```

Теперь вы можете найти ID пользователя, вставленного с помощью `mysql_insert_id`, и сохранить его в переменной. Затем вы можете получить ID изображения, используя `mysql_insert_id` еще раз. И наконец, вы можете обновить содержимое столбца `profile_pic_id` в строке нового пользователя в таблице `users`. Это сработало бы, и вы в конечном итоге пришли бы к трем различным операциям взаимодействия с базой данных.

1. `INSERT` для помещения информации о пользователе в таблицу `users`.
2. `INSERT` для помещения изображения в таблицу `images`.
3. `UPDATE` для вставки ID нового изображения в таблицу `users`.

На данный момент эти три действия могут показаться не слишком значительными, но каждое взаимодействие с вашей базой данных отнимает время и ресурсы. Согласно общему принципу ваше взаимодействие с базой данных должно сводиться к минимально необходимому количеству раз. Это не говорит о том, что вы не должны работать с базой данных, вы просто не должны совершать три или четыре запроса, если можно выполнить ту же задачу за один или два запроса.

В данном случае можно сократить количество взаимодействий с базой данных MySQL с трех до двух.

1. `INSERT` для вставки изображения в таблицу `images` (и для получения в процессе этой вставки ID этого изображения).
2. `INSERT` для вставки нового пользователя в таблицу `users` и применения только что полученного ID изображения в качестве части данных, помещаемых в эту инструкцию `INSERT`.

И чего мы этим добьемся? Переход от трех взаимодействий с MySQL к двум звучит, возможно, не слишком впечатляюще. И все же вы сократили взаимодействие с базой данных на треть! Если можно сделать меньше запросов, то так и нужно делать.

Необходимо продолжить работу и поменять местами инструкции INSERT:

```
// Получение данных изображения

$insert_image_sql = sprintf("INSERT INTO images " .
                                  "(filename, mime_type, " .
                                   "file_size, image_data) " .
                            "VALUES ('%s', '%s', %d, '%s');",
                            mysql_real_escape_string($image_filename),
                            mysql_real_escape_string($image_mime_type),
                            mysql_real_escape_string($image_size),
                            mysql_real_escape_string($image_data));

mysql_query($insert_image_sql);

// Этот код заменяет прежнее присваивание значения переменной $insert_sql
$insert_sql = sprintf("INSERT INTO users " .
                             "(first_name, last_name, email, " .
                              "bio, facebook_url, twitter_handle) " .
                      "VALUES ('%s', '%s', '%s', '%s', '%s', '%s');",
                      mysql_real_escape_string($first_name),
                      mysql_real_escape_string($last_name),
                      mysql_real_escape_string($email),
                      mysql_real_escape_string($bio),
                      mysql_real_escape_string($facebook_url),
                      mysql_real_escape_string($twitter_handle));

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . $user_id);
exit();
```

ПРИМЕЧАНИЕ

Здесь нет какого-либо дополнительного кода. Это просто полная перестановка создания вставки и перемещение вызова функции mysql_query, который связан с пользователем, из места перед кодом, связанным с изображением, в место после этого кода.

Но вы можете удалить часть кода. Теперь, когда вставка в таблицу users производится во вторую очередь, вы можете вернуться к использованию в перенаправлении функции mysql_insert_id.

Теперь осталось только получить ID из инструкции INSERT, связанной со вставкой изображения, и воспользоваться этим идентификатором в инструкции INSERT, связанной со вставкой пользователя в таблицу users. Но вы уже знаете, как это сделать: можно применить функцию mysql_insert_id для получения ID той строки,

которая вставлена в таблицу images, а затем добавить его к вашей инструкции INSERT для таблицы users.

```
// Получение данных изображения

$insert_image_sql = sprintf("INSERT INTO images " .
                                "(filename, mime_type, " .
                                 "file_size, image_data) " .
                            "VALUES ('%s', '%s', %d, '%s');",
                            mysql_real_escape_string($image_filename),
                            mysql_real_escape_string($image_mime_type),
                            mysql_real_escape_string($image_size),
                            mysql_real_escape_string($image_data));

mysql_query($insert_image_sql);

// Этот код заменяет прежнее присваивание значения переменной $insert_sql
$insert_sql = sprintf("INSERT INTO users " .
                          "(first_name, last_name, email, " .
                           "bio, facebook_url, twitter_handle, " .
                           "profile_pic_id) " .
                      "VALUES ('%s', '%s', '%s', '%s', '%s', '%s', %d);",
                      mysql_real_escape_string($first_name),
                      mysql_real_escape_string($last_name),
                      mysql_real_escape_string($email),
                      mysql_real_escape_string($bio),
                      mysql_real_escape_string($facebook_url),
                      mysql_real_escape_string($twitter_handle),
                      mysql_insert_id());

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . mysql_insert_id());
exit();
```

ПРИМЕЧАНИЕ

Следует помнить, что, поскольку ID изображения, вставляемого в profile_pic_id, является целым числом, а не строкой, в качестве спецификатора типа для функции sprintf следует использовать %d и вам не нужно заключать это значение в одинарные кавычки.

Соберите все вместе. Обновленная версия create_user.php должна принять следующий вид:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';

$upload_dir = SITE_ROOT . "uploads/profile_pics/";
```

```
$image_fieldname = "user_pic";

// Потенциальные PHP-ошибки отправки файлов
$php_errors = array(1 => 'Превышен макс. размер файла, указанный в php.ini',
                2 => 'Превышен макс. размер файла, указанный в форме HTML',
                3 => 'Была отправлена только часть файла',
                4 => 'Файл для отправки не был выбран.');

$first_name = trim($_REQUEST['first_name']);
$last_name = trim($_REQUEST['last_name']);
$email = trim($_REQUEST['email']);
$bio = trim($_REQUEST['bio']);
$facebook_url = str_replace("facebook.org", "facebook.com", trim($_
REQUEST['facebook_url']));
$position = strpos($facebook_url, "facebook.com");
if ($position === false) {
  $facebook_url = "http://www.facebook.com/" . $facebook_url;
}

$twitter_handle = trim($_REQUEST['twitter_handle']);
$twitter_url = "http://www.twitter.com/";
$position = strpos($twitter_handle, "@");
if ($position === false) {
  $twitter_url = $twitter_url . $twitter_handle;
} else {
  $twitter_url = $twitter_url . substr($twitter_handle, $position + 1);
}

// Проверка отсутствия ошибки при отправке изображения
($_FILES[$image_fieldname]['error'] == 0)
  or handle_error("сервер не может получить выбранное вами изображение.",
                  $php_errors[$_FILES[$image_fieldname]['error']]);

// Является ли этот файл результатом нормальной отправки?
@is_uploaded_file($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы попытались совершить безнравственный поступок. Позор!",
                  "Запрос на отправку: файл назывался " .
                  "'{$_FILES[$image_fieldname]['tmp_name']}'");

// Действительно ли это изображение?
@getimagesize($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы выбрали файл для своего фото, " .
                  "который не является изображением.",
                  "{$_FILES[$image_fieldname]['tmp_name']} " .
                  "не является файлом изображения.");

// Присваивание файлу уникального имени
```

```
$now = time();
while (file_exists($upload_filename = $upload_dir . $now .
                                      '-' .
                   $_FILES[$image_fieldname]['name'])) {
    $now++;
}

// Вставка изображения в таблицу images
$image = $_FILES[$image_fieldname];
$image_filename = $image['name'];
$image_info = getimagesize($image['tmp_name']);
$image_mime_type = $image_info['mime'];
$image_size = $image['size'];
$image_data = file_get_contents($image['tmp_name']);

$insert_image_sql = sprintf("INSERT INTO images " .
                            "(filename, mime_type, file_size, image_data) " .
                            "VALUES ('%s', '%s', %d, '%s');",
                            mysql_real_escape_string($image_filename),
                            mysql_real_escape_string($image_mime_type),
                            mysql_real_escape_string($image_size),
                            mysql_real_escape_string($image_data));

mysql_query($insert_image_sql);

$insert_sql = sprintf("INSERT INTO users " .
                             "(first_name, last_name, email, " .
                              "bio, facebook_url, twitter_handle, " .
                              "profile_pic_id) " .
                      "VALUES ('%s', '%s', '%s', '%s', '%s', '%s', %d);",
                      mysql_real_escape_string($first_name),
                      mysql_real_escape_string($last_name),
                      mysql_real_escape_string($email),
                      mysql_real_escape_string($bio),
                      mysql_real_escape_string($facebook_url),
                      mysql_real_escape_string($twitter_handle),
                      mysql_insert_id());

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . mysql_insert_id());
exit();
?>
```

Проверьте работу кода путем создания еще одного пользователя. Затем посмотрите, каким будет последний и самый большой вставленный ID изображения из вашей таблицы images:

```
mysql> select image_id from images;
+----------+
| image_id |
+----------+
|        4 |
|        5 |
|        6 |
+----------+
3 rows in set (0.00 sec)
```

Он должен быть таким же, как ID, который был вставлен в запись последнего пользователя, помещенного в таблицу users:

```
mysql> select user_id, first_name, last_name, profile_pic_id from users;
+---------+------------+-----------+----------------+
| user_id | first_name | last_name | profile_pic_id |
+---------+------------+-----------+----------------+
|       1 | C. J.      | Wilson    |           NULL |
|       5 | Peter      | Gabriel   |           NULL |
|       7 | Bob        | Jones     |           NULL |
|      22 | James      | Roday     |           NULL |
|      30 | William    | Shatner   |              6 |
+---------+------------+-----------+----------------+
7 rows in set (0.01 sec)
```

Отлично! Теперь вы можете увидеть, что при вставке изображения ID этого изображения попадает в таблицу users и вы получаете связь между пользователем и изображением.

Связывание таблиц с помощью условия WHERE

Как же получить изображение при наличии пользователя? Все начинается с ID пользователя. По этому идентификатору можно выбрать нужного вам пользователя:

```
$select_query = sprintf("SELECT * FROM users WHERE user_id = %d",
                        $user_id);
```

Это просто sprintf-версия кода из show_user.php. Внесите соответствующие изменения в свою собственную версию show_user.php.

Но теперь вы получаете не только информацию о пользователе, но и profile_pic_id для этого пользователя. Следовательно, вы можете применить данный идентификатор для получения изображения для этого пользователя:

```
if ($result) {
  $row = mysql_fetch_array($result);
  $first_name     = $row['first_name'];
  $last_name      = $row['last_name'];
```

```
  $bio            = preg_replace("/[\r\n]+/", "</p><p>", $row['bio']);
  $email          = $row['email'];
  $facebook_url   = $row['facebook_url'];
  $twitter_handle = $row['twitter_handle'];
  $profile_pic_id = $row['profile_pic_id'];

  $image_query = sprintf("SELECT * FROM images WHERE image_id = %d",
                         $profile_pic_id);
  $image_result = mysql_query($image_query);

  // Превращение $twitter_handle в URL
  $twitter_url = "http://www.twitter.com/" .
                     substr($twitter_handle, $position + 1);

}
```

ПРИМЕЧАНИЕ

Вы можете удалить из show_user.php код, который относится к путевому имени файлов изображений профилей, поскольку вам теперь не нужно использовать этот подход для работы с изображениями.

Код работает, но получается, что один потенциальный шаг превращается в два. Здесь устанавливается связь между двумя таблицами: связующим звеном является часть информации — profile_pic_id в таблице users и image_id в таблице images.

Связывание таблиц с помощью общих столбцов

Вы также приобрели способ получения только конкретных строк из таблицы: условие WHERE.

Чтобы все это объединить, можно получить пользователя из таблицы users и изображения из таблицы images, где profile_pic_id пользователя соответствует image_id изображения:

```
SELECT first_name, last_name, filename
  FROM users, images
 WHERE profile_pic_id = image_id;
```

Запустите этот код в окне командной строки MySQL и увидите следующий результат:

```
mysql> SELECT first_name, last_name, filename
    ->   FROM users, images
    ->  WHERE profile_pic_id = image_id;
+------------+-----------+---------------------------+
| first_name | last_name | filename                  |
+------------+-----------+---------------------------+
| William    | Shatner   | 220px-William_Shatner.jpeg |
+------------+-----------+---------------------------+
1 row in set (0.02 sec)
```

ВНИМАНИЕ

Пока вы не вставите в свою базу данных сведения о человеке по имени William Shatner, ожидать появления точно таких же данные не придется.

Очень интересный результат! Вам удалось связать вместе две таблицы. В одном запросе объединилась информация из одной таблицы с соответствующей информацией из другой таблицы. Великолепно.

Использование псевдонимов для таблиц (и столбцов)

Но несмотря на всю эффективность данного запроса, в нем есть все-таки небольшая путаница. Взглянем на код еще раз:

```
SELECT first_name, last_name, filename
  FROM users, images
 WHERE profile_pic_id = image_id;
```

Вполне очевидно, что first_name и last_name являются столбцами таблицы users. Но если вам неизвестна структура базы данных, трудно сразу понять, откуда берется столбец filename. (Разумеется, вам-то отлично известна ваша база данных, и вы знаете, что filename является столбцом таблицы images.)

То же самое касается profile_pic_id и image_id. Это имена столбцов, но к какой таблице относится каждый из столбцов?

Эту ситуацию можно прояснить, воспользовавшись для столбцов префиксами в виде имен их таблиц. Этому запросу можно придать более очевидную форму:

```
SELECT users.first_name, users.last_name, images.filename
  FROM users, images
 WHERE users.profile_pic_id = images.image_id;
```

Будет получен такой же результат, но сам запрос при этом утратит свою неопределенность. А теперь нужно учесть еще один немаловажный фактор: программисты по своей природе слишком ленивы. Большинство программистов лучше наберут один-два символа, если при этом можно будет избежать набора пяти или десяти символов. И SQL вполне поддерживает особенность их психологии. Таблице можно присвоить псевдоним, поставив после ее имени одну-две буквы, а затем во всем остальном запросе воспользоваться этими буквами в качестве префикса:

```
SELECT u.first_name, u.last_name, i.filename
  FROM users u, images i
 WHERE u.profile_pic_id = i.image_id;
```

Функционально этот запрос ничем не отличается от предыдущего запроса, но при этом он стал более понятным и кратким, что наилучшим образом отвечает потребностям программиста.

Покажите мне изображение

На данный момент вы располагаете всеми данными и даже можете получить изображение для конкретного пользователя. Осталось только показать это изображение, не так ли?

Так и есть, но теперь ситуация в корне отличается от той, когда изображение уже имелось в файловой системе и нужно было только указать на файл. В данном случае вы должны загрузить необработанные данные изображения из своей базы данных, а затем каким-то образом дать понять браузеру: «Это не текст, а изображение. Покажи его». Это в принципе нетрудно, но очень сильно отличается от того, что вы делали раньше.

Вывод изображения

Сначала нужно создать сценарий, который сможет загрузить и показать изображение. После его создания нужно будет просто сослаться на него из сценария show_user.php. Поэтому данный сценарий является важным звеном, состоящим из совершенно нового кода.

Создадим новый сценарий и назовем его show_image.php. Для начала можно взять за основу обычную оболочку сценария, присутствующую во всех имеющихся у вас сценариях:

```
<?php

require '../scripts/app_config.php';
require '../scripts/database_connection.php';

?>
```

Стратегия сценария

Теперь нужно определить все, что должно происходить в этом сценарии. Вы можете обозначить основные шаги.

1. Получить из запроса ID изображения.
2. Создать SELECT-запрос к таблице images, используя ID изображения.
3. Запустить SELECT-запрос и получить результаты.
4. Взять из результатов только то, что должно быть строкой.
5. Сообщить браузеру, что он должен получить изображение.
6. Сообщить браузеру о том, изображение какого типа он должен получить.
7. Передать браузеру данные изображения.

За исключением нескольких последних шагов, вы уже, наверное, забежали далеко вперед в представлениях о том коде, который нужно создать. Но нужно будет также позаботиться и об обработке множества ошибок, которые могут произойти в процессе выполнения этого кода.

1. Нужно убедиться, что ID изображения было отправлено сценарию.
2. Необходимо убедиться, что ID отображается на изображение в таблице images.
3. Следует справиться с общими проблемами, возникающими при загрузке или отображении данных изображения.

Ни один из этих пунктов также не отличается особой сложностью реализации. Следовательно, настало время приступить к работе.

Получение ID изображения

В первую очередь вам нужно получить ID для загрузки изображения из базы данных. Попутно можно приступить к обработке ошибки: если в качестве части запроса не поступит ID, произойдет сбой.

```
<?php

require '../scripts/app_config.php';
require '../scripts/database_connection.php';

if (!isset($_REQUEST['image_id'])) {
  handle_error("не указано изображение для загрузки.");
}

$image_id = $_REQUEST['image_id'];

?>
```

Все довольно просто и очень похоже на код, который создавался ранее в `show_user.php`. И опять-таки функция `handle_error` превращает обработку возникающих проблем в сущие пустяки.

Создание и запуск SELECT-запроса

Теперь для создания SQL-запроса можно воспользоваться услугами вашего нового друга `sprintf`, а для получения результата — более старого друга `mysql_query`.

```
<?php

// Инструкции require

// Получение ID изображения

// Создание инструкции SELECT
$select_query = sprintf("SELECT * FROM images WHERE image_id = %d",
                        $image_id);
// Запуск запроса
$result = mysql_query($select_query);

?>
```

Вот, собственно, и все.

Получение результатов, получение изображения и обработка потенциальных ошибок

Теперь из переменной `$result` можно извлечь нужные данные. Ранее вы уже делали это несколько раз. Вы осуществляли последовательный перебор всех строк, возвращенных запросом:

```
  if ($return_rows) {
    // Мы располагаем строками из запроса, предназначенными для показа
    echo "<p>Результаты, возвращенные запросом:</p>";
    echo "<ul>";
    while ($row = mysql_fetch_row($result)) {
      echo "<li>{$row[0]}</li>";
    }
    echo "</ul>";
  } else {
    // Строк нет. Сообщение о том, был запущен запрос или нет.
    echo "<p>Следующий запрос был обработан успешно:</p>";
    echo "<p>{$query_text}</p>";
  }
```

ПРИМЕЧАНИЕ

Этот код был взят из главы 5. Просто невероятно, насколько за несколько коротких глав усложнились ваши сценарии.

Вы также пользовались инструкцией if, если ожидался только один результат:

```
if ($result) {
  $row = mysql_fetch_array($result);

  // Обработка единственного результата
} else {
  handle_error("возникла проблема с поиском вашей " .
               "информации в нашей системе.",
               "Ошибка обнаружения пользователя с ID {$user_id}");
}
```

В этой инструкции предполагается, что если значение переменной $result эквивалентно true, то вы получили строку. Далее она просто игнорирует любые строки, кроме первой, зная, что сгенерированный SQL-запрос может вернуть только одну строку.

В сценарии show_image.php вам нужно что-либо подобное последнему подходу. Но можно будет провести проверку и убедиться в наличии результата и без заключения всего кода в if-структуру:

```
<?php

// Инструкции require
// Получение ID изображения
// Создание и запуск запроса

if (mysql_num_rows($result) == 0) {
  handle_error("запрошенное изображение найти невозможно.",
               "Не найдено изображение с ID" . $image_id . ".");
}

$image = mysql_fetch_array($result);

?>
```

Этот подход более нагляден, поскольку при нем продолжается выполнение кода после обработки ошибки. (Почему такая последовательность считается более естественной, объяснено в следующей врезке.)

ЗАМЕТКИ О ПРОЕКТИРОВАНИИ

Последовательный код обычно более понятен

В программировании почти всегда одну и ту же задачу можно решить более чем одним способом. Обычно есть сразу несколько хороших способов выполнения того или иного задания. Но, как правило, существует наиболее понятный способ, к которому и нужно стремиться. Вам нужен качественный, работоспособный код, в котором также можно будет легко разобраться.

Но создание понятного кода становится труднее по мере того, как этот код становится сложнее. Зачастую имеется сразу несколько точек принятия решения (с `if`-инструкциями), обработок ошибок, циклов и других всевозможных конструкций, повсеместно встречающихся в коде. Из-за всей этой сложности вы стремитесь сделать свой код как можно более последовательным. Иными словами, вы хотите получить возможность читать этот код более-менее от начала и до конца и не терять при этом нить развития событий.

Посмотрите с этой точки зрения еще раз на раннюю версию сценария `show_user.php`:

```
if ($result) {
  $row = mysql_fetch_array($result);
  // Обработка единственного результата
} else {
   handle_error(
               "возникла проблема с поиском вашей " .
               "информации в нашей системе.",
               "Ошибка обнаружения пользователя с ID
               {$user_id}");
}
```

Этот код работает, и он достаточно надежен. Но можно ли считать его последовательным?

В какой-то степени да. Если есть результат, его нужно получить и обработать. Если его нет, необходимо заняться ошибками. Однако как вы сами считаете? Как выглядит настоящая последовательность?

Сначала нужно посмотреть, есть ли результат, и, если его нет, обработать ошибку. Затем только после того, как вы убедитесь в безопасности продолжения, нужно обработать результаты и продолжить выполнение программы. Следовательно, с данной точки зрения все, что находится за обработкой ошибки, выпадает из последовательности. Это то, что вы концептуально хотели бы сделать до перехода к работе со строкой таблицы.

Поэтому обновленная последовательность в `show_image.php`, где сначала идет обработка ошибок, а затем используется результат, является более подходящим решением с точки зрения читаемости кода. При сохранении той же функциональности данный код проще в понимании и в поддержке.

Сообщение браузеру о характере поступающих данных

Теперь у вас есть информация, которую вы хотели получить об изображении, но пока вы не можете доставить ее браузеру. То есть можете, но браузер, скорее всего, в ней запутается. Он привык иметь дело с HTML, а не с необработанными двоичными данными или с чем-нибудь еще.

Браузеру нужно сообщить следующие сведения.

- Содержимое какого типа ему поступит. Такая информация передается браузеру посредством MIME-типа и обычно выглядит как text/html или text/xml, а в случае с использованием изображений она может принимать вид image/jpeg, image/gif или image/png.
- Каким будет размер поступающей информации, если тип относится к двоичным (как в случае с изображениями). Браузерам это нужно знать, чтобы определить момент окончания поступления информации.

Как бы это ни показалось вам удивительным, у вас уже есть средство общения с браузером, позволяющее выполнить именно эту задачу. Помните такую строку кода:

```
header("Location: " . SITE_ROOT . "scripts/show_error.php?" .
        error_message={$user_error_message}&" .
        system_error_message={$system_error_message}");
```

Этот код напрямую общается с браузером. Он отправляет браузеру заголовок, который называется Location. Значением этого заголовка является место, то есть URL, и браузер знает, что при получении заголовка Location нужно перейти на указанный URL, который является значением заголовка.

Следовательно, функция header в PHP является механизмом, с помощью которого можно непосредственно общаться с браузером. Что касается двух информационных элементов, которые вам нужно отправить (тип содержимого и размер этого содержимого), для обоих этих элементов у браузера есть специальные заголовки:

- Content-type — позволяет вам сообщить браузеру, содержимое какого MIME-типа вы собираетесь отправить;
- Content-length — дает вам возможность задать размер, то есть «длину» файла той информации, которую вы собираетесь отправить.

На данный момент оба этих элемента информации у вас имеются в таблице images, в столбцах mime_type и file_size.

Соберите все вместе, и вы получите две строки кода, которые нужно добавить в сценарий show_image.php:

```
<?php

// Инструкции require
// Получение ID изображения
// Создание и запуск запроса
// Получение результата и обработка ошибок в случае его отсутствия

header('Content-type: ' . $image['mime_type']);
header('Content-length: ' . $image['file_size']);

?>
```

Теперь браузер ожидает вполне определенный тип информации (в вашем случае image/jpeg или в большинстве случаев — image/gif). Он знает размер информации, и теперь ему нужно лишь получить саму информацию.

Отправка данных изображения

Что еще осталось сделать? Да самый простой из шагов:

```
<?php

// Инструкции require
// Получение ID изображения
// Создание и запуск запроса
// Получение результата и обработка ошибок в случае его отсутствия
// Сообщение браузеру с помощью заголовков и поступающей информации

echo $image['image_data'];

?>
```

Теперь эти данные не являются строкой текста, они представляют собой обычную двоичную информацию, извлеченную из блоб-столбца таблицы images и передаваемую побитно. Но магия кроется не в этой строке. Она состоит в том, что вы сообщили браузеру о конкретной разновидности информации и о конкретном ее размере. Благодаря этому браузер знает: «Поступает изображение, и именно в этом качестве нужно рассматривать приходящую информацию».

Перехват и обработка ошибок

Итак, все запланированное для показа изображения уже выполнено:

1. Из запроса получено ID изображения.
2. Создан SELECT-запрос к таблице images с использованием ID изображения.
3. Запущен SELECT-запрос и получены результаты.
4. Из результатов взято только то, что должно быть строкой.
5. Браузеру сообщено, что он должен получить изображение.
6. Браузеру сообщено о том, изображение какого типа он должен получить.
7. Браузеру переданы данные изображения.

Отлично. При этом сценарий достаточно краток, понятен и легко читаем. Победа по всем статьям.

Вы также позаботились об обработке большинства ошибок.

1. Убедились, что сценарию был передан ID изображения.
2. Убедились, что ID отображается на изображение в таблице images.
3. Разобрались с общими проблемами, возникающими при загрузке или показе данных изображения.

Но это все же еще не соответствует действительности. Первые две задачи выполнены, а как насчет так называемых общих проблем? Что будет, если, к примеру, произойдет ошибка при отправке заголовка Content-type? Или при отправке заголовка Content-length? А что можно сказать о показе изображения? Может ли при этом случиться какой-нибудь сбой? Что если данные изображения будут испорчены, случится что-нибудь при извлечении данных из набора, полученного в результате запроса, или браузер не сможет обработать тип изображения, которое попытается отправить ваш сценарий?

Во всех этих случаях вы получите неучтенную ошибку. При общих ошибках такого сорта, выходящих за рамки однозначно толкуемых, заранее проверяемых и позволяющих удостовериться в отсутствии проблем, вам придется с ними разбираться.

Проблема в том, что вы не можете их отловить. Вам нужен способ сказать: «Если при работе этого блока кода случится что-нибудь, имеющее общий характер, нужно сделать следующее...». И это «сделать следующее» у вас есть в handle_error. PHP предоставляет способ выполнить это с помощью блока try-catch.

Часть try блока try-catch является блоком, куда помещается весь код, при выполнении которого может произойти ошибка. Этим вы словно говорите: «Попробуй выполнить этот код». Часть catch блока try-catch запускается только при возникновении ошибки. То есть если в какой-то момент времени внутри блока try что-то пойдет не так, запустится часть catch этого блока.

Но это еще не все, коду catch передается объект Exception. Этот объект обладает информацией о том, что именно пошло не так, следовательно, вы можете составить отчет, например, для такой пользовательской функции, как handle_error.

Чтобы расположить этот код в show_image.php, сначала нужно поместить весь код, при выполнении которого может произойти ошибка, в try и фигурные скобки:

```
<?php

require '../scripts/app_config.php';
require '../scripts/database_connection.php';

try {
  if (!isset($_REQUEST['image_id'])) {
    handle_error("не указано изображение для загрузки.");
  }

  $image_id = $_REQUEST['image_id'];

  // Создание инструкции SELECT
  $select_query = sprintf("SELECT * FROM images WHERE image_id = %d",
                          $image_id);

  // Запуск запроса
  $result = mysql_query($select_query);

  if (mysql_num_rows($result) == 0) {
```

```
    handle_error("запрошенное изображение невозможно найти.",
                 "Не найдено изображение с ID " . $image_id . ".");
  }

  $image = mysql_fetch_array($result);

  header('Content-type: ' . $image['mime_type']);
  header('Content-length: ' . $image['file_size']);

  echo $image['image_data'];
}
?>
```

Теперь весь этот код закрыт. Если что-нибудь пойдет не так, интерпретатор PHP выдаст объект `Exception` (исключение), сообщающий о характере проблемы, и перейдет к выполнению блока `catch`:

```
<?php

require '../scripts/app_config.php';
require '../scripts/database_connection.php';

try {
  // код, при выполнении которого может произойти ошибка
} catch (Exception exc) {
}
?>
```

Можно заметить, что этот код очень похож на функцию: когда управление передается коду `catch`, он получает также и объект `Exception`. При этом `exc` является именем переменной исключения, следовательно, при необходимости вы можете ссылаться на это исключение.

ПРИМЕЧАНИЕ

Имя переменной exc не начинается с символа $, потому что для объектов в PHP отсутствие префикса $ в именах считается вполне обычным явлением. Вы еще узнаете об объектах PHP много интересного, но сейчас нужно понять, что exc — это переменная, но ее тип отличается от тех типов, с которыми вам приходилось работать до сих пор.

И наконец, вы должны сделать в этом блоке `catch` что-нибудь полезное:

```
<?php

require '../scripts/app_config.php';
require '../scripts/database_connection.php';

try {
  // код, при выполнении которого может произойти ошибка
} catch (Exception exc) {
  handle_error("при загрузке вашего изображения произошел сбой.",
               "Ошибка при загрузке изображения: " . $exc->getMessage());
}
?>
```

В этот коде для вас не должно быть никаких сюрпризов, за исключением, может быть, странных символов ->. Когда происходят ошибки, на помощь приходит функция handle_error. Как обычно, функции handle_error передается понятная пользователю строка, а также некоторая дополнительная информация для программистов, которые могут отслеживать работу сценария. В таком случае сообщение исходит от переменной exc и метода getMessage. Объект в PHP не имеет функций, он имеет методы. А ссылка на метод осуществляется с помощью символов ->.

ПРИМЕЧАНИЕ

Если вы здесь ничего не поняли, не стоит волноваться. Вскоре вы будете разбираться в объектах вполне профессионально, а сейчас просто примите код как он есть и поверьте, что очень быстро все встанет на свои места.

Итак, при выполнении этого кода выдается отчет о любой ошибке, которая может произойти, и пресекаются попытки PHP продолжить выполнение блока try.

В show_image.php должен быть следующий код:

```
<?php

require '../scripts/app_config.php';
require '../scripts/database_connection.php';

try {
  if (!isset($_REQUEST['image_id'])) {
    handle_error("не указано изображение для загрузки.");
  }

  $image_id = $_REQUEST['image_id'];

  // Создание инструкции SELECT
  $select_query = sprintf("SELECT * FROM images WHERE image_id = %d",
                          $image_id);

  // Запуск запроса
  $result = mysql_query($select_query);

  if (mysql_num_rows($result) == 0) {
    handle_error("запрошенное изображение найти невозможно.",
                 "Не найдено изображение с ID " . $image_id . ".");
  }

  $image = mysql_fetch_array($result);

  header('Content-type: ' . $image['mime_type']);
  header('Content-length: ' . $image['file_size']);

  echo $image['image_data'];
} catch (Exception $exc) {
  handle_error("при загрузке вашего изображения произошел сбой.",
               "Ошибка при загрузке изображения: " . $exc->getMessage());
}
?>
```

Итак, что же нам еще осталось сделать? Провести несколько тестов, чтобы убедиться в работоспособности кода.

Тест, тест и еще раз тест

Откройте окно командной строки MySQL и найдите вставленное изображение. Заметьте, какой у него ID:

```
mysql> select image_id, filename from images;
+----------+---------------------------+
| image_id | filename                  |
+----------+---------------------------+
|        6 | 220px-William_Shatner.jpeg |
+----------+---------------------------+
1 row in set (0.03 sec)
```

Теперь откройте браузер и наберите URL для show_image.php. Но не нажимайте клавишу Enter! Вы, конечно, можете это сделать, но при этом получите ошибку, сообщение о которой показано на рис. 9.4, поскольку вы не предоставили ID.

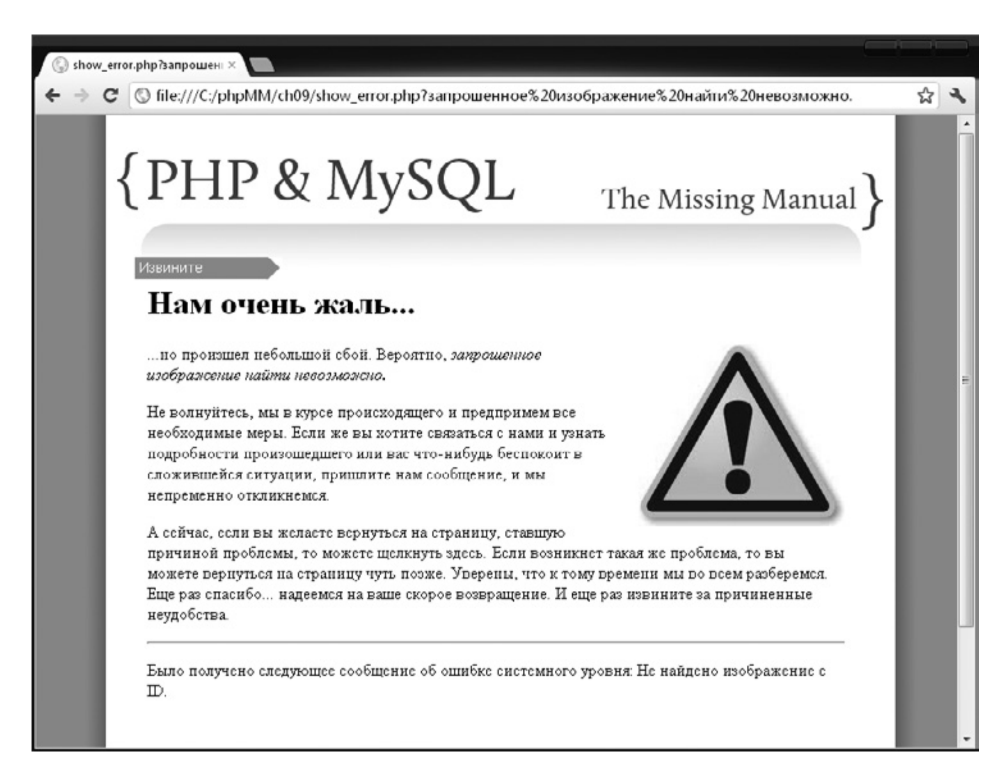


Рис. 9.4. Не указан ID изображения

А теперь добавьте к URL идентификатор изображения: show_image.php?image_id=6. Поместите это в адресную строку (наряду со своим доменным именем и путем), и вы должны получить результат, показанный на рис. 9.5.

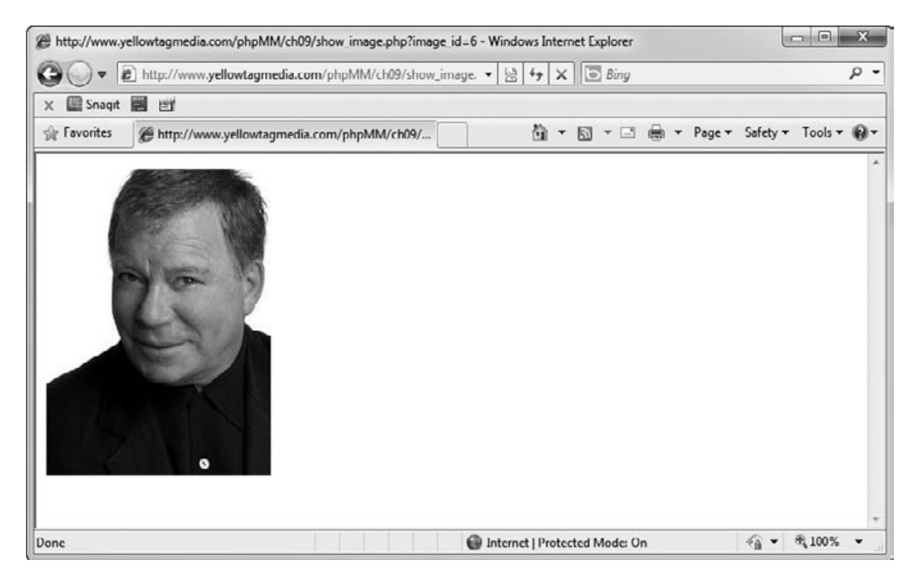


Рис. 9.5. Вывод изображения

Для этого и проделывалась вся ваша работа: нужно было заставить браузер показать изображение. Это похоже на результат щелчка правой кнопкой мыши на изображении какой-нибудь другой веб-страницы и выбора пункта контекстного меню Открыть картинку в новой вкладке. Показано только изображение без какого-либо сопроводительного текста.

Итак, теперь вы получили в качестве своего изображения звезду экрана Уильяма Шетнера, и это уже можно считать успехом.

Встроить изображение ничуть не сложнее, чем его просмотреть

И наконец, вернемся к сценарию show_user.php. Вспомним, что на самом деле show_image.php был неким обходным вариантом. Он был необходим, но нашей целью не был сценарий, показывающий изображение. Целью был сценарий, показывающий данные пользователя, а это значит, что он должен показывать и изображение этого пользователя. Сейчас вся работа, необходимая для его полноценного функционирования, завершена. Следовательно, в show_user.php теперь можно собрать весь код.

Вам нужен лишь идентификатор изображения

Первое, что могло бы прийти вам в голову, — переписать SQL-запрос, который извлекает запись из таблицы images на основе пользователя из таблицы users:

```
SELECT u.first_name, u.last_name, i.filename

  FROM users u, images i
 WHERE u.profile_pic_id = i.image_id;
```

Но нужно ли это делать? Нет, не нужно, поскольку все, что требуется show_image.php, — это ID изображения, и этот идентификатор есть в таблице users в столбце profile_pic_id. Вам не нужно связывать таблицы users и images.

Поэтому при получении результатов вашего SQL-запроса вам необходимо взять ID изображения профиля:

```
<?php

require '../scripts/app_config.php';
require '../scripts/database_connection.php';

// Получение ID отображаемого пользователя
// Создание инструкции SELECT
// Запуск запроса

if ($result) {
  $row = mysql_fetch_array($result);
  $first_name     = $row['first_name'];
  $last_name      = $row['last_name'];
  $bio            = preg_replace("/[\r\n]+/", "</p><p>", $row['bio']);
  $email          = $row['email'];
  $facebook_url   = $row['facebook_url'];
  $twitter_handle = $row['twitter_handle'];
  $image_id       = $row['profile_pic_id'];

  // Превращение $twitter_handle в URL
} else {
  handle_error("возникла проблема с поиском вашей информации в нашей системе.",
               "Ошибка обнаружения пользователя с ID {$user_id}");
}
?>

<!-- HTML -->
```

ПРИМЕЧАНИЕ

Строка с новым кодом заменяет старую строку, где извлекался URL изображения в той версии, где в вашей таблице users хранился только путь к изображению.

Сценарий может быть в виде указания в теге источника изображения (src)

Имея данный идентификатор, вы готовы к работе с потерянным изображением. Но то, что произойдет дальше, может показаться немного странным, и поэтому требующим в свою очередь некоторых объяснений.

Подумайте о стандартном простом HTML-элементе img:

```
<img src="/images/roday.jpg" />
```

Его часть img сообщает браузеру, что нужно ожидать изображение. А его атрибут src сообщает браузеру место, где находится изображение. Но указание этого места

является намерением осуществления еще одного запроса со стороны браузера — в данном случае запроса файла /images/roday.jpg. А что получает браузер из указания этого места? Пакет битов, составляющий изображение roday.jpg.

Но в самом этом месте, будь то roday.jpg или его URL, нет ничего магического. Это всего лишь место, и поскольку из него браузеру возвращается изображение, осуществляется показ этого изображения. Следовательно, вполне допустимо предоставить в качестве значения атрибута src все, что угодно, лишь бы это значение приводило к возврату изображения. Вы можете предоставить, скажем, сценарий, который выводит изображение. Вы можете даже вручить браузеру это изображение следующим образом:

```
<img src="show_image.php?image_id=6" />
```

И поскольку show_image.php с правильным ID возвращает изображение, браузер покажет это изображение в том месте, которое занимает на вашей странице тег img.

В результате вы можете изменить свой HTML, чтобы при его выполнении все именно так и происходило:

```
<?php
  // Большой объем полезного PHP-кода
?>
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Профиль</div>

  <div id="content">
    <div class="user_profile">
      <h1><?php echo "{$first_name} {$last_name}"; ?></h1>
      <p><img src="show_image.php?image_id=<?php echo $image_id; ?>"
              class="user_pic" />
        <?php echo $bio; ?></p>
      <p class="contact_info">
        Поддерживайте связь с <?php echo $first_name; ?>:
      </p>
      <ul>
        <!-- Контактные ссылки -->
      </ul>
    </div>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

Вот и все! Теперь атрибут src тега img указывает на ваш сценарий с правильным ID. Поэтому, когда вы соберете вместе весь код сценария show_user.php, должно получиться примерно следующее:

```
<?php

require '../scripts/app_config.php';
require '../scripts/database_connection.php';

// Получение ID отображаемого пользователя
$user_id = $_REQUEST['user_id'];

// Создание инструкции SELECT
$select_query = sprintf("SELECT * FROM users WHERE user_id = %d",
                        $user_id);

// Запуск запроса
$result = mysql_query($select_query);

if ($result) {
  $row = mysql_fetch_array($result);
  $first_name     = $row['first_name'];
  $last_name      = $row['last_name'];
  $bio            = preg_replace("/[\r\n]+/", "</p><p>", $row['bio']);
  $email          = $row['email'];
  $facebook_url   = $row['facebook_url'];
  $twitter_handle = $row['twitter_handle'];
  $image_id       = $row['profile_pic_id'];

  // Превращение $twitter_handle в URL
  $twitter_url = "http://www.twitter.com/" .
                 substr($twitter_handle, $position + 1);
} else {
  handle_error("возникла проблема с поиском вашей информации в нашей системе.",
               "Ошибка обнаружения пользователя с ID {$user_id}");
}
?>

<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Профиль</div>

  <div id="content">
    <div class="user_profile">
      <h1><?php echo "{$first_name} {$last_name}"; ?></h1>
```

```
      <p><img src="show_image.php?image_id=<?php echo $image_id; ?>"
              class="user_pic" />
        <?php echo $bio; ?></p>
      <p class="contact_info">
        Поддерживайте связь с <?php echo $first_name; ?>:
      </p>
      <ul>
        <li>... по электронной почте
          <a href="<?php echo $email; ?>"><?php echo $email; ?></a></li>
        <li>... путем
          <a href="<?php echo $facebook_url; ?>"> посещения его страницы
             в Facebook </a></li>
        <li>... путем <a href="<?php echo $twitter_url; ?>"> отслеживания
             его сообщений в Twitter</a></li>
      </ul>
    </div>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

Результаты показаны на рис. 9.6.



Рис. 9.6. Результат выполнения кода

Отличная работа. Кто бы мог подумать, что перед тем, как перелистнуть последнюю страницу этой главы, вы сможете самостоятельно загружать биты и байты из базы данных и отображать их по требованию в виде изображения?

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Нет ли какого-нибудь интерактивного руководства по работе с изображениями?

Если вы проводите много времени в Интернете, то, наверное, знаете, насколько эффективна поисковая система Google. Потратьте пять минут на поиск в этой системе, и вы найдете минимум 20 или 30 руководств по отправке изображений на сервер средствами PHP, касающихся как хранения путей к изображениям в вашей базе данных, так и хранения в ней самих изображений. Там даже есть целые программные структуры, предоставляющие вам весь необходимый программный код!

Тогда зачем же нужно было пробираться через все эти хитросплетения PHP, которые вы только что преодолели просто для того, чтобы сделать все это самостоятельно? Есть две очень важные причины, почему код такого сорта не только важен для набора в вашем текстовом редакторе, но также для понимания всего в нем происходящего.

Во-первых, используя программные структуры, которые вы найдете в Интернете, вы сможете сделать очень многое. И, нужно сказать правду, многие из этих программных структур, особенно если брать их из авторитетных источников, делают то, что должен делать ваш код, намного лучше, быстрее и эффективнее. Но это еще не означает, что вам не нужно понимать, что, собственно, происходит. На самом деле, осознавая принципы работы этого кода, вы будете намного лучше подготовлены к наилучшему выбору используемой программной структуры и к пониманию того, почему именно она может быть лучше написанного вами кода. Конечно, это произойдет только после того, как вы напишете данный код и будете готовы для перехода к использованию более совершенного кода.

И во-вторых, по мере написания все большего количества веб-приложений вы все чаще будете понимать, что ваши запросы становятся все более и более определенными. То есть вам понадобится отправка изображения на сервер, но она будет нужна вам с некими конкретными особенностями или поправками, присущими вашему приложению. Возможно, потребуется принимать изображения только в формате JPG, а не GIF, или необходимо будет навязать ограничения по размеру, диктуемые серверной стороной, а не руководствоваться полем ввода HTML, устанавливающим максимальный размер.

Но если вы не понимаете принципов работы этой разновидности кода, то как вы сможете вносить подобные поправки? Будь то ваш код или код, созданный кем-то другим, вы должны уметь вносить подобные коррективы, связанные с индивидуальными настройками и специализацией кодового фрагмента. Для этого нужны знания, которые приходят только в результате получения собственной практики разработки программ.

Итак, какой же подход лучше?

Итак, у вас есть два абсолютно разных подхода к получению пользовательских изображений на основе информации, хранящейся в базе данных (как минимум путей к таким изображениям). Вы потратили много времени, работая над данным кодом, как, собственно, и над любым другим кодом в процессе вашего путешествия по PHP. Теперь нужно ответить на вопрос: какой подход лучше?

Самым подходящим ответом будет следующий: «Все зависит от обстоятельств». Или: «Вы сами должны это решить». Возможно, эти ответы вас расстроят и никоим образом не удовлетворят. Причина в том, что на вопросы такого сорта, касающиеся хранения изображений, обработки ошибок или взаимодействия с другими программными структурами и кодом, созданным другими людьми, не всегда можно получить однозначно «правильные» ответы.

Насколько велика по объему та файловая система, с которой нужно работать? Зависит ли сумма ваших платежей от объема файлов вашего веб-сервера? Увеличивается или уменьшается сумма платежей в зависимости от платежей, начисляемых за размер вашей базы данных? Доступна ли ваша база данных в локальном режиме и насколько быстро осуществляется доступ? Нет ли какого-нибудь медленного соединения с другой машиной? Ваш ответ зависит от всех этих обстоятельств.

И все же в конце рабочего дня иногда вы вынуждены сказать: «Я не уверен, вроде этот подход лучше... или тот». И это вполне нормально. Вы можете выбрать тот или иной подход, попробовать, как он работает, и не останавливаться на этом выборе. Есть много случаев, когда единственно *неверным* решением будет затягивание с анализом вариантов на несколько часов (дней, недель!) вместо того, чтобы двигаться вперед.

Ладно, если вы непременно хотите получить конкретный ответ...

Если вы в чем-то сомневаетесь, храните свои изображения на файловом сервере, а в базе данных держите только пути к этим изображениям. Создать хороший код, который и сохраняет изображение в базе данных, и показывает это изображение, нетрудно, куда сложнее выбрать правильный вариант. При каждом запуске `SELECT`-запроса на получение ваших изображений из таблицы `images` и извлечении содержимого столбца `image_data` вы выбираете весь размер данных этого изображения. Предположим, у вас есть 100 строк, в каждой из которых хранится изображение со средним размером 1 Мбайт. Тогда получается 100 Мбайт данных изображений, загружающих ваш сетевой трафик и канал связи с базой данных.

Поэтому при наличии сомнений остановитесь на хранении в вашей базе данных простого пути к изображению. Но зачем же тогда нужно было читать целую главу, посвященную альтернативному подходу? А затем, что теперь у вас есть общее

понятие о том, что происходит с изображениями, независимо от того, хранятся они в базе данных или нет.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

А как же тогда привести свою базу данных к прежнему виду?

При прочих равных условиях работа с изображениями, хранящимися в файловой системе, представляется более удачным решением. (Но, по правде говоря, равных условий просто не бывает!) Однако поскольку этот вариант вполне подходит в качестве исходного, следующие примеры предполагают, что вы настроились именно на него. Как же вернуться к этому решению?

Во-первых, вы должны были сделать резервную копию своих сценариев. Если вы не сделали этого, то можете заново загрузить примеры файлов с веб-страницы с материалами компакт-дисков данной книги и воспользоваться версией, в которой изображения не хранятся в базе данных.

Во-вторых, вам нужно будет удалить столбец `profile_pic_id` из вашей таблицы `users`. Вот код SQL, вносящий это изменение:

```
ALTER TABLE users
      DROP COLUMN profile_pic_id;
```

Затем вы можете довольно легко удалить таблицу `images`:

```
DROP TABLE images;
```

Вот и все. Вы вернулись к прежнему режиму работы.

10 Вывод списков, итерация и администрирование

Долгое время основное внимание уделялось общим деталям: пользователю, его информации и расширению этой информации, картинке его профиля. Все это было вполне оправданно. Вы достаточно близко познакомились с PHP и MySQL, выяснили, что существует не один, а два способа решения наиболее общей проблемы PHP: загрузки изображений, и вам удалось при этом сохранить хороший внешний вид всего, с чем вы работали. И это нельзя считать какими-то малозначительными достижениями, их нужно оценивать как весьма впечатляющие.

Но все же это был весьма узконаправленный взгляд. Теперь же в качестве пользователя вы можете настроить и определить содержимое некой основной информации. А что если вы не находитесь в роли пользователя? Если вам нужно увидеть, сколько пользователей в вашей системе, вам нужно удалить какого-нибудь злоумышленника, обновить картинку, поскольку она неприемлема из этических соображений, вам приходится делать все это с помощью окна командной строки MySQL. В этот нет ничего плохого, и вы, конечно же, способны на это. Но вы уже, наверное, поняли, что в этом большом и суровом мире веб-приложений большинство администраторов не держат в углу своих экранов запущенный терминал MySQL.

Вместо этого у них есть интерфейсы администрирования. Используя эти интерфейсы, они могут вывести список всех пользователей, находящихся в системе. Здесь они могут расставить в разных местах флажки и удалить сразу нескольких пользователей. Они могут просмотреть данные любого нужного им пользователя. И все это делается в простом и понятном веб-интерфейсе. Вы можете придать вашему веб-приложению точно такой же удобный интерфейс.

Думаю, что веб-приложение, где пользователи предоставляют свой основной социальный профиль, не собирается в скором времени вытеснить Facebook, Twitter или Google+. Но приступая к обдумыванию интерфейса администрирования, вы сталкиваетесь со всевозможными интересными проблемами. Вы должны использовать различные типы SQL-запросов, смешивать более существенные объемы кода PHP и MySQL со своим HTML, поскольку вам придется выводить в списке сведения о каждом пользователе из базы данных. Вам придется работать с инструкциями удаления — `DELETE` и с множеством условных инструкций `WHERE`.

Иными словами, будет необходимо задействовать весь арсенал своих знаний и продвигаться дальше. Вам не понадобится большое количество радикально новых технологий, но существует множество важных вариаций того, что вы уже знаете. Итак, зачем откладывать все это в долгий ящик или мириться с использованием терминала MySQL в качестве интерфейса администрирования? Настало время получить в конце концов более наглядный способ, позволяющий не отставать от своих пользователей.

Вещи, которые никогда не меняются

С чего же начать? А с того же, с чего начинались практически все другие задачи: выяснения необходимого и набросков общих планов того, как все это должно выглядеть и взаимодействовать. Можно начать с нескольких пунктов, определяющих необходимые экраны, и с создания наряду с этим нескольких макетов либо в HTML, либо даже с использованием таких инструментов, как Photoshop.

Поскольку наше приложение не отличается особой сложностью, на данный момент можно довольствоваться весьма небольшим перечнем:

- форма, в которой выводится список всех пользователей системы;
- ссылка на страницу профиля каждого пользователя;
- возможность удаления пользователя;
- возможность обновления или изменения информации о пользователе;
- средства, дающие другим пользователям административные привилегии.

Последний пункт потребует, наверное, выполнения существенного объема работы и создания некоторых уникальных вещей, для чего придется преодолеть массу сложностей, поэтому мы его отложим на потом (до следующей главы). А вот все остальное нам вполне по силам.

Пользовательский интерфейс, или Краткость по-прежнему сестра таланта

Теперь вы можете создать сложную систему страниц, позволяющую вам управлять всеми этими взаимодействиями. Сценарий `show_user.php` может определить, являетесь ли вы администратором, и выборочно показывать кнопку Удалить. Вы можете создать целое меню администрирования. Здесь уместно еще раз напомнить, что иногда чем проще, тем лучше. Кроме того, нужно взять за общее правило, что чем меньше щелчков, задействующих сеть, тем лучше. Если вы можете предоставить единственную страницу, вмещающую в себя все самые востребованные функции, то, наверное, вы сможете обойтись только этой одной страницей.

В данном случае именно это вы и можете сделать. Можно вывести список пользователей в простой последовательности, превратить имя каждого пользователя

в ссылку на страницу его профиля и даже добавить кнопку удаления напротив каждого пользователя. Вы должны будете продумать вопрос изменения информации о пользователе, но все же три функции в одной форме являются неплохим началом.

Итак, на что же это должно быть похоже? На рис. 10.1 показана неплохая отправная точка. Конечно, этот дизайн вряд ли завоюет какие-нибудь награды. Нужно лучше выровнять элементы удаления, неважно выглядят используемые по умолчанию небольшие маркеры списка, да и в целом эта страница нуждается в серьезной доработке. Но многое можно позже исправить. А в данный момент она дает некое представление, необходимое для начала работы. Таким на данном этапе и должен быть макет: отправной точкой и прототипом.



Рис. 10.1. Прототип страницы администрирования

В HTML-коде для этой страницы сразу видно множество дубликатов, количество которых можно уменьшить с помощью PHP:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пользователи</div>

  <div id="content">
   <ul>
```

```
    <li>
     <a href="show_user.php?user_id=30">Уильям Шетнер</a>
     (<a href="mailto:bill@williamshatner.com">bill@williamshatner.com</a>)
     <a href="delete_user.php?user_id=30">
       <img class="delete_user" src="../images/delete.png" width="15" />
     </a>
    </li>
    <li>
     <a href="show_user.php?user_id=22">Джеймс Родей</a>
     (<a href="mailto:james@roday.net">james@roday.net</a>)
     <a href="delete_user.php?user_id=22">
       <img class="delete_user" src="../images/delete.png" width="15" />
     </a>
    </li>
    <li>
     <a href="show_user.php?user_id=1">К. Дж. Вильсон</a>
     (<a href="mailto:cj@texasrangers.com">cj@texasrangers.com</a>)
     <a href="delete_user.php?user_id=1">
       <img class="delete_user" src="../images/delete.png" width="15" />
     </a>
    </li>
   </ul>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

Нужен также список пожеланий

До сих пор вы переходили от макета непосредственно к коду. В целом, конечно, неплохо, но это означает, что при создании макета в коде все, что к нему добавляется, является загадкой. Будет ли все это хорошо работать, учитывая способ, применяемый при создании ваших страниц и сценариев? Или же вам придется все перепроектировать, чтобы ваши новые идеи вписались в существующую программную среду?

Наверное, макетам можно было уделить довольно много времени. Можно было бы потратить большое количество времени, чтобы нарисовать в Photoshop небольшие красные крестики справа и расставить интервалы. Конечно же, ничто в HTML и CSS не сможет выглядеть похожим на макет, созданный в Photoshop, но можно получить приблизительно ту же картину.

А проблемы между тем удвоятся. Во-первых, будет потрачено много времени на внешний вид, прежде чем будет создан какой-нибудь код. Во-вторых, вы даже не будете задумываться о том, как решения, принятые для реализации существующего кода, могут повлиять на будущие решения и функциональные возможности. То есть вы создаете код без всякого реального предвидения.

Каков же выход? Нужно просто составить краткий список будущих функциональных возможностей, которые вы надеетесь реализовать. В нем не должно

быть никаких фантазий, простой текстовый документ или стикеры (для гибкого стиля разработки) или какие-нибудь заметки в iPad или iPhone, которые потом перекочуют на ваш рабочий компьютер, когда все удачно сложится. Дополняйте или обновляйте этот список по мере работы и изменения свойств и функциональных возможностей. Будем надеяться, что наличие этой «следующей версии» будущих удобств поможет вам подумать о том, как те решения, которые принимаются сегодня, могут помочь или навредить вам в том случае, когда придется создавать дополнительный код завтра, на следующей неделе или в следующем месяце.

Сейчас при наличии основных функциональных свойств было бы неплохо добавить следующее.

- Разработать более привлекательный пользовательский интерфейс, выровнять различные «столбцы» данных, придав им более наглядный вид, а также выровнять по вертикали кнопки удаления в виде крестиков.
- Добавить в пользовательские профили изображения, чтобы иметь в интерфейсе администратора более наглядное графическое представление о каждом пользователе.
- Ввести возможность выбора и удаления на одном экране сразу нескольких пользователей.
- Добавить диалоговое окно подтверждения удаления пользователя, чтобы избежать случайных удалений.

Вы должны добавить к этому списку собственные идеи, но и в этом виде он уже является неплохой отправной точкой. Независимо от того, будете вы воплощать все задуманное в программном коде или нет, теперь по крайней мере вы сможете принимать решения, идущие на пользу, а не во вред разработке вашего приложения.

ПРИМЕЧАНИЕ

Иногда независимо от того, насколько хорошо составлено перспективное планирование, текущие свойства требуют от вас принимать решения, которые могут впоследствии затруднить реализацию свойств, находящихся в списке пожеланий. Это в порядке вещей. Намного важнее обеспечить своевременную готовность первостепенных свойств.

Вывод списка всех пользователей

Сначала нужно решить первостепенные задачи: перед тем как получить возможность добавления кнопок удаления и изображений профилей, а также выравнивания элементов, вам нужен список всех ваших пользователей. Что касается SQL-запроса, который для этого нужно составить, данная задача не составляет особого труда. Можно, например, сделать следующее:

```
SELECT *
  FROM users;
```

Но это немного грубоватый подход. Его нужно улучшить, чтобы повысить производительность, придать коду более понятный вид, и вообще проявить хорошие знания PHP и MySQL. В первую очередь нужно сделать самое важное: придать запросу нужную форму.

Выбор с помощью SELECT нужной (на данный момент) информации

Инструкция SELECT * проводит выборку всего, что есть в таблице. Если вы устанавливали связи между таблицами, эта инструкция выбирает все из всех таблиц, с которыми установлены связи. На данный момент, что касается таблицы users, повода для особых тревог нет. Вот как выглядят все столбцы, которые вы собираетесь извлечь с помощью инструкции SELECT *:

```
mysql> describe users;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| user_id        | int(11)      |      | PRI | NULL    | auto_increment |
| first_name     | varchar(20)  |      |     |         |                |
| last_name      | varchar(30)  |      |     |         |                |
| email          | varchar(50)  |      |     |         |                |
| facebook_url   | varchar(100) | YES  |     | NULL    |                |
| twitter_handle | varchar(20)  | YES  |     | NULL    |                |
| bio            | text         | YES  |     | NULL    |                |
| user_pic_path  | varchar(200) | YES  |     | NULL    |                |
| profile_pic_id | int(11)      | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
9 rows in set (0.10 sec)
```

ПРИМЕЧАНИЕ

В зависимости от того, насколько вы следовали всем указаниям, у вас может иметься столбец user_pic_path и может не быть столбца profile_pic_id. Возможно, именно в таком состоянии вы и хотели бы видеть свою базу данных, поэтому вам не нужно беспокоиться о внешнем ключе с таблицей images, которую вы больше не используете.

Вы можете избавиться от этого столбца с помощью следующей инструкции:

```
ALTER TABLE users
    DROP COLUMN profile_pic_id;
```

Но взгляните еще раз на рис. 10.1. Вам нужна не вся эта информация, а только лишь first_name, last_name и user_id для гиперссылки на show_user.php, а также email пользователя. Значит, инструкция SELECT * извлекает несколько ненужных столбцов: facebook_url, twitter_handle, bio и user_pic_path.

А какое это имеет значение? Нарушается важный принцип: извлекать только строго необходимую информацию.

При каждом выборе всех записей из таблицы users вы получаете еще одну строку. И извлечение каждого столбца в этих строках занимает определенное время, сокращая полосу пропускания вашей сети и отнимая ресурсы. Предположим, у вас имеется 100, 1000 или 10 000 пользователей, а также допустим, что каждый из них написал биографию, занимающую 20 абзацев. В таком случае, просто отказавшись от выборки всей информации с помощью символа * (а следовательно, и от выборки столбца bio) из таблицы users, вы сэкономите большой объем трафика и потребляемых ресурсов.

Итак, что же вам сейчас нужно? Всего лишь несколько столбцов:

```
SELECT user_id, first_name, last_name, email
  FROM users;
```

И больше ничего. Именно это вы должны выбрать с помощью инструкции SELECT.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

А должен ли я также выбрать то, что понадобится позже?

В будущем было бы неплохо добавить на страницу администрирования изображения пользовательских профилей. Вам уже известно, что в столбце user_pic_path таблицы users содержится путь к таким изображениям. Стало быть, если вы в конечном счете хотите его заполучить, нужно ли уже сейчас выбирать этот столбец с помощью инструкции SELECT?

С одной стороны, хорошо иметь инструкцию SELECT, уже настроенную на будущее свойство, которое вы непременно хотите получить. А с другой стороны, пока это свойство не реализовано, поэтому хотите ли вы на самом деле получить код, выполняющий что-то только наполовину? К тому же пока не похоже, что вы собираетесь создавать код для отображения изображения пользовательского профиля. Это касается только лишь наличия данных к тому моменту, кода этот код будет создаваться.

Конечно, нужно задумываться об отношениях того, что вы сейчас создаете, к будущим свойствам. Однако следует фокусироваться на создании того кода, который решает не будущие, а текущие проблемы. Подумайте о том, насколько скользкой может быть такая дорожка. Вы можете начать с выбора биографии, поскольку когда-нибудь вам захочется иметь на странице администрирования выборку из нее. Вы можете пойти еще дальше и выбрать социальную информацию для создания дополнительных ссылок для контакта с пользователем. И так незаметно вы вернетесь к инструкции SELECT * и к способу, при котором берется больше информации, чем в конечном итоге используется.

Хорошо, что вы уже знаете, что добавить извлечение пользовательского изображения будет совсем нетрудно, когда придет время. Необходимо будет просто изменить содержимое инструкции SELECT. Но сейчас следует остановиться и сфокусироваться на написании кода для существующей работы. Оставьте будущую работу на потом.

Есть и еще одна причина поступить именно таким образом: в какой-то момент вашей программистской карьеры вам необходимо будет приступить к оценке своей работы. Вы должны будете знать, сколько времени (в часах или днях) займет реализация той или иной функциональной составляющей. Обычно вы выставляете счет за свою работу, который по крайней мере частично основывается на таких оценках, поэтому важно оценить все как можно точнее. Если вы начнете смешивать текущие и будущие функциональные компоненты, ваши оценки потеряют смысл. Все закончится тем, что вы запросите чрезмерную цену или, что еще хуже, слишком дешево оцените свой труд, поскольку не будете заниматься поэтапным созданием приложения.

Создание простой страницы администрирования

Итак, теперь у вас есть подходящая инструкция `SELECT`. Настало время создать еще один сценарий. Но перед тем как это сделать, нужно принять еще одно важное решение: как назвать этот сценарий? Возможно, ему вполне подойдет имя `admin.php`, но насколько продуманным будет такой выбор?

Посмотрите на названия других используемых вами сценариев:

- `create_user.php` — создает нового пользователя;
- `show_user.php` — показывает пользователя по заданному идентификатору;
- `app_config.php` — конфигурирует приложение;
- `database_connection.php` — осуществляет подключение к базе данных.

Каждое из этих имен описывает действие сценария. Это очень удобно, поскольку сразу становится понятно, как использовать тот или иной сценарий и даже как эти сценарии могут взаимодействовать. Например, `create_user.php` создает пользователя, а затем должен, наверное, передать управление сценарию `show_user.php`.

А чем занимается сценарий `admin.php`? Что если вам со временем понадобится добавить форму и сценарий, которые позволят администратору изменить пароль пользователя? Это работа администратора, но она не имеет отношения к сценарию `admin.php`. То же самое справедливо и по отношению добавления пользователя к группе или к обновлению полей в форме. Все это «администрирование», но ни одна из этих задач не задействует данный сценарий.

По сути, этот сценарий выводит список всех пользователей. Если применить ту же схему присваивания имен, которая использовалась для других сценариев, то лучше дайте ему более описательное имя `show_users.php`.

Итак, откройте новый файл, назовите его `show_users.php` и начните с выбора только той информации о всех пользователях, которая вам нужна:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';

// Создание инструкции SELECT
$select_users =
  "SELECT user_id, first_name, last_name, email " .
  "  FROM users";

// Запуск запроса
$result = mysql_query($select_users);
?>
```

ПРИМЕЧАНИЕ

Поскольку никакие вставки в запрос SELECT не делаются, нет смысла использовать функцию sprintf. Можно просто создать запрос в виде строки.

Можно также пойти дальше и настроить «оболочку» HTML-страницы: те части, которые заведомо не будут генерироваться вашим сценарием:

```
<?php
// Получение всех пользователей
?>

<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пользователи</div>

  <div id="content">
   <ul>
     <!-- Сюда в <li>-теги будут попадать все пользователи -->
   </ul>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

Пока здесь особо смотреть не на что, но вы все же можете протестировать код и убедиться в отсутствии ошибок в вашем PHP или HTML. На рис. 10.2 показана пустая, но не имеющая ошибок страница, полученная в результате выполнения сценария show_users.php. Тестирование каждой стадии вашей разработки займет

всего несколько минут, а при создании нового сценария роль тестирования трудно переоценить!

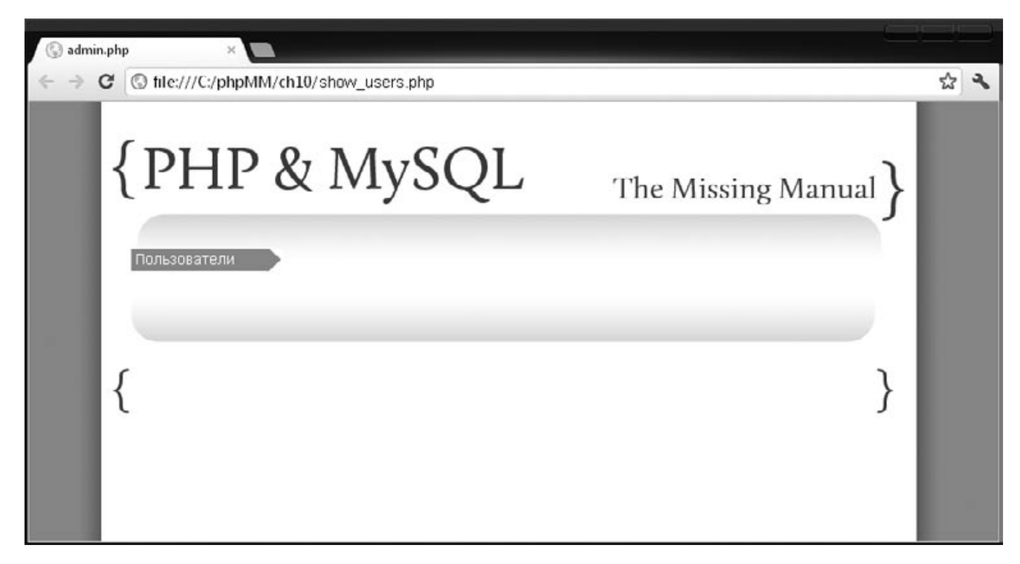


Рис. 10.2. Здесь вы можете убедиться в отсутствии ошибок, которые могли бы произойти при подключении к вашей базе данных или при выполнении вашей инструкции SELECT

Перебор элементов массива

Теперь нужно заполнить тег <li> для каждого пользователя. Всю необходимую строку HTML можно создать, применив еще раз функцию sprintf:

```
$user_row = sprintf(
  "<li><a href='show_user.php?user_id=%d'>%s %s</a> " .
  "(<a href='mailto:%s'>%s</a>) " .
  "<a href='delete_user.php?user_id=%d'><img " .
     "class='delete_user' src='../images/delete.png' " .
     "width='15' /></a></li>",
  // информация для заполнения значений
  );
```

ПРИМЕЧАНИЕ

Здесь нет особых преимуществ в использовании sprintf по сравнению с применением кавычек и фигурных скобок с переменными внутри. Но как только вы начинаете задействовать sprintf, вырабатывается привычка использования этой функции везде, где нужно вставлять значения переменных в строки. Эта функция становится своеобразным стандартным и к тому же весьма удобным средством.

Это довольно длинная строка, но в конечном итоге она должна приобрести примерно следующий вид:

```
<li><a href='show_user.php?user_id=1'>К.Дж. Вильсон</a>
  (<a href='mailto:cj@texasrangers.com'>cj@texasrangers.com</a>)
```

```
   <a href='delete_user.php?user_id=1'><img class='delete_user'
      src='../images/delete.png' width='15' /></a></li>
```

Теперь вам нужно лишь осуществить циклический перебор результатов запроса. Это совсем несложно, вы уже делали это, применяя следующий код:

```
while ($row = mysql_fetch_row($result)) {
  echo "<li>{$row[0]}</li>";
}
```

А затем, конечно же, можно с помощью следующего кода получить каждую часть данных, возвращенных в результате запроса:

```
while ($row = mysql_fetch_row($result)) {
  echo "<li>{$row['col_name']}</li>";
}
```

С помощью этого кода извлекается конкретное значение из массива `$row`, связанное с именем столбца `col_name`.

Если применить это конкретно к вашей таблице `users` и к столбцам, о возвращении которых известно, а затем вставить в HTML, получится следующее:

```
<?php
  // Получение всех пользователей
?>
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пользователи</div>

  <div id="content">
   <ul>
     <?php
       while ($user = mysql_fetch_array($result)) {
         $user_row = sprintf(
           "<li><a href='show_user.php?user_id=%d'>%s %s</a> " .
           "(<a href='mailto:%s'>%s</a>) " .
           "<a href='delete_user.php?user_id=%d'><img " .
              "class='delete_user' src='../images/delete.png' " .
              "width='15' /></a></li>",
           $user['user_id'], $user['first_name'], $user['last_name'],
           $user['email'], $user['email'], $user['user_id']);
         echo $user_row;
       }
     ?>
   </ul>
  </div>
  <div id="footer"></div>
</body>
</html>
```

ПРИМЕЧАНИЕ

В этом HTML есть ссылка на сценарий, который еще не написан: delete_user.php. Ничего страшного, скоро мы и до него доберемся. Здесь программирование ведется с учетом другой работы, ожидающей своего завершения.

На первый взгляд эта длинная функция `sprintf` может показаться похожей на какие-то переходы на новые строки. Но при более пристальном рассмотрении можно заметить, что это просто слишком длинная отдельная строка HTML, которая получается в результате объединения всех составляющих. Если разобраться, то здесь нет ничего особо сложного.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Возрастающая беспорядочность вашего HTML начинает вызывать опасения

По мере написания все более и более сложного кода происходит что-то неуловимое. Сначала ваши сценарии состоят практически из одного PHP, и в них, возможно, используется команда `echo` для вывода нескольких строк текста. Затем вы начинаете создавать сценарии, имеющие блок PHP-кода в самом начале, а затем какую-то часть PHP-кода в самом конце. Затем появляются сценарии, где то в одном, то в другом месте встречаются небольшие вставки PHP-кода в код HTML, и так до конца сценария.

Теперь у вас есть сценарий `show_users.php`. В нем имеется блок PHP-кода, затем идет код HTML... а затем он превращается в какую-то сплошную мешанину из PHP, который осуществляет вывод довольно больших фрагментов HTML. Сейчас вы могли бы, наверное, написать тот же фрагмент вывода, который выдает HTML, а затем содержит множество небольших фрагментов PHP, вставленных то тут, то там, но в принципе это не меняет общей картины. Неважно, как вы все это перекроите, у вас все равно получится мешанина из HTML и PHP.

И здесь вы сталкиваетесь с реальными опасениями, связанными с программированием на PHP: с часто возникающей склонностью к получению в конечном итоге смеси из кода и разметки.

Как только начинается подобная мешанина, граница между кодом и представлением (разметкой, показывающей что-нибудь вашему пользователю) становится слишком тонкой, если и вовсе не исчезает. Вставить где-нибудь посередине кода HTML большой блок PHP довольно просто. Но просто еще не означает хорошо. По мере возможностей старайтесь все же располагать основную часть своего кода PHP в начале сценария, а данные вставлять только при необходимости.

Дело продвигается, и вы готовы посмотреть, как все это выглядит. Внесите усовершенствования в `show_users.php` и убедитесь, что все, от чего зависит этот сценарий, находится на своих местах. На рис. 10.3 показано все, к чему вы стремитесь.

Хотя увиденное все еще нельзя отнести к произведениям искусства, но все же это существенный шаг вперед. Щелкните на записи любого пользователя и убедитесь в том, что вы попадаете на правильную страницу `show_user.php`, создаваемую для этого пользователя (рис. 10.4).

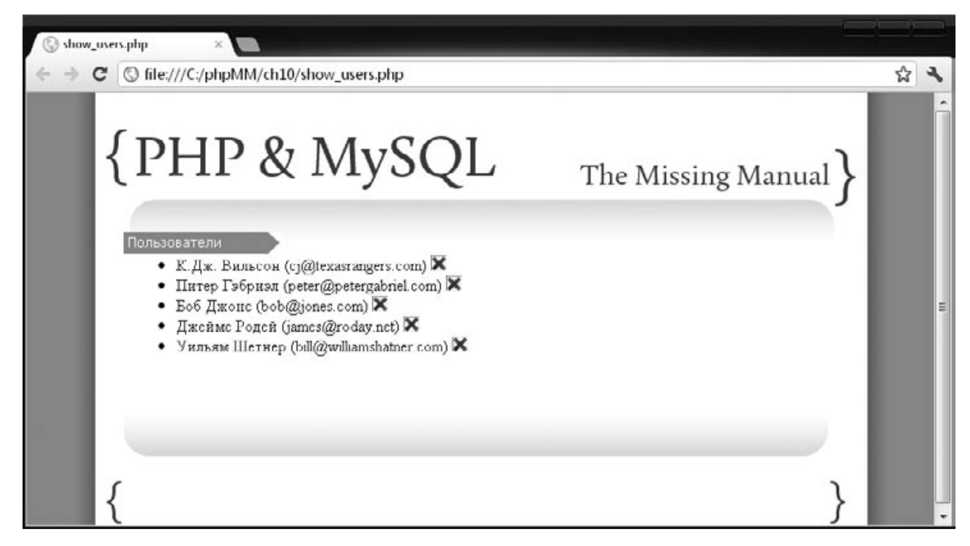


Рис. 10.3. Результат усовершенствования сценария

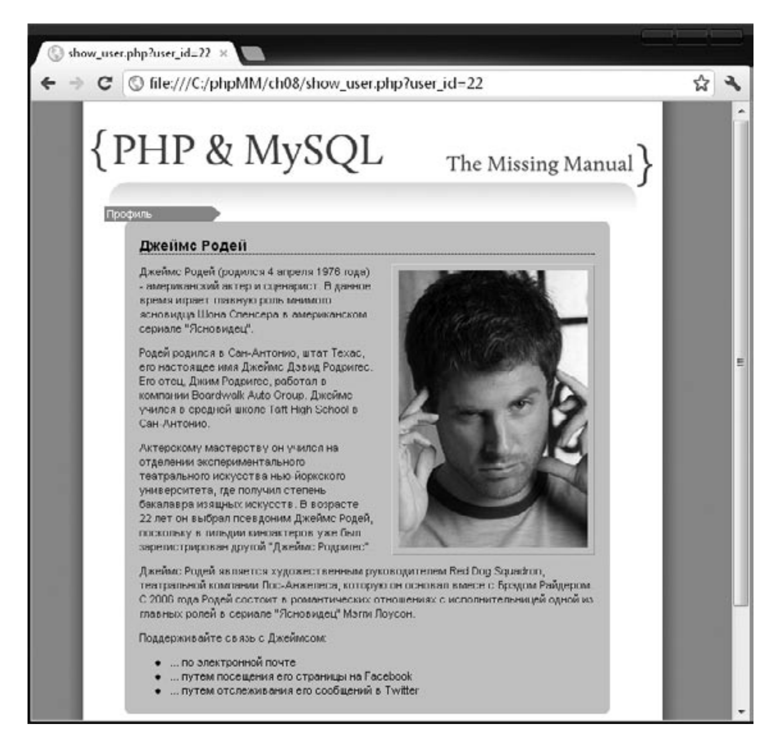


Рис. 10.4. Страница show_user.php, созданная для пользователя

Вы, наверное, уже привыкаете к сценариям, вызывающим другие сценарии, в которых, в свою очередь, создаются ссылки на другие сценарии. Поскольку в своей карьере PHP-программиста вы собираетесь пользоваться этим довольно часто,

торопиться не нужно. Верите вы этому или нет, существуют крупномасштабные PHP-приложения, которые вообще не используют никаких HTML-файлов в чистом виде. Например, Wordpress на 100 % состоит из PHP.

Удаление пользователя

В программировании часто новые задачи приводят к новым испытаниям, к необходимости освоения новых технологий, усвоения новых свойств языка программирования. Все это, конечно, интересно, но может также и портить настроение. Темп работы зачастую замедляется как минимум на несколько часов, а иногда и на несколько дней, пока не станут ощущаться результаты.

Затем в вашем арсенале появляются приемы, знания и умения для выполнения новой задачи. К одному из таких простых примеров можно отнести и удаление пользователя.

Разбор отдельных компонентов

Как выглядит запрос на удаление пользователя? Вам уже знакома следующая инструкция:

```
DELETE FROM users;
```

Чтобы сосредоточиться на конкретном пользователе, к ней нужно добавить условие WHERE:

```
DELETE FROM users
      WHERE user_id = [id_конкретного_пользователя];
```

Здесь нет ничего нового. Идентификатор пользователя user_id можно взять из того сценария, который вызывает ваш сценарий. Именно это уже делается в сценарии show_users.php:

```
<?php
  while ($user = mysql_fetch_array($result)) {
    $user_row = sprintf(
      "<li><a href='show_user.php?user_id=%d'>%s %s</a> " .
      "(<a href='mailto:%s'>%s</a>) " .
      "<a href='delete_user.php?user_id=%d'><img " .
         "class='delete_user' src='../images/delete.png' " .
         "width='15' /></a></li>",
      $user['user_id'], $user['first_name'], $user['last_name'],
      $user['email'], $user['email'], $user['user_id']);
    echo $user_row;
  }
?>
```

Как только этот код превратится в HTML, вы получите следующую строку кода:

```
<a href='delete_user.php?user_id=22'>...</a>
```

Этот код должен быть очень похож на то, что вы уже делали раньше, когда отправляли user_id сценарию show_user.php:

```
// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . mysql_insert_id());
```

ПРИМЕЧАНИЕ

Данный код был в сценарии create_user.php. Пользователь перенаправлялся на страницу просмотра после того, как его информация сохранялась в базе данных.

И после того, как вы добрались до user_id и удалили пользователя, вы можете просто вернуться назад к сценарию show_users.php, который повторно с помощью инструкции SELECT проведет выборку информации о пользователях, и удаленный пользователь просто исчезнет. Отлично!

Объединение всех составляющих

Теперь осталось только перенабрать различные фрагменты из ваших других сценариев и внести изменения в некоторых местах. В результате получится delete_user.php, имеющий следующий вид:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';

// Получение идентификатора удаляемого пользователя
$user_id = $_REQUEST['user_id'];

// Создание инструкции DELETE
$delete_query = sprintf("DELETE FROM users WHERE user_id = %d",
                        $user_id);

// Удаление пользователя из базы данных
mysql_query($delete_query);

// Перенаправление на show_users для повторного показа пользователей
// (без удаленного пользователя)
header("Location: show_users.php");
exit();
?>
```

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Настоящие программисты вырезают и вставляют код

Только что вы создали первый сценарий, почти полностью составленный из повторно используемого ранее написанного кода. Но этот код не подходит для применения в `app_config.php`. Поэтому в данном случае вам не нужно то там, то здесь выделять какие-то фрагменты кода и помещать их в сервисные функции, как это делалось при обработке ошибок или настройке подключений к базе данных.

Если к данному времени вы прочитали множество книг по программированию, то уже, наверное, приготовились к следующему замечанию: не нужно заниматься вырезкой и вставкой кода! Подобные действия наносят вред, вырезка и вставка приводят к досадным, трудно обнаруживаемым ошибкам, к ожирению и к расстройству личной жизни. Так ведь? Именно это говорят авторы компьютерных книг.

Конечно, все это просто нелепо. Общеизвестно, что чрезмерное потребление коктейля «Маргарита» и шоколадного мусса могут стать причиной лишних пяти килограммов. Но вы, наверное, знаете, что несмотря на все предостережения, каждый программист, который занимается программированием по многу часов в день, знает сочетания клавиш для копирования, вырезки и вставки и часто ими пользуется. В конце концов, если программирование вошло в их жизнь, то им наверняка известны сочетания клавиш не только в одной, но и в других системах, а также клавиши, выполняющие те же функции в редакторах emacs и vi и в любых других редакторах, которые когда-либо приходилось применять. Это ключевые моменты их работы.

Тогда к чему все эти грозные предостережения? Согласен, некоторые трудно отслеживаемые ошибки вызываются вырезкой, копированием и вставкой кода. Кроме того, в результате бывают и некоторые несоответствия. В одном фрагменте копируемого кода переменная называется `$insert_sql`, а в другом — `$insert_query`. Это приводит к сбоям, PHP не всегда справляется с уведомлениями о таких проблемах, и распутывать подобный клубок приходится самостоятельно. Но эта проблема не имеет отношения к копированию и вставке, она относится к вопросам непоследовательности в присваивании имен переменным.

Настоящие предостережения (точнее, некоторые из них) можно сформулировать следующим образом.

- Следует знать о повышенном риске при копировании, вырезке и вставке. Следовательно, нужно все делать внимательно и без лишней спешки.
- По возможности необходимо сократить количество источников, из которых ведется вырезка и копирование. Тогда будет меньше вероятности столкнуться с проблемами пар имен переменных и тому подобными неприятностями.
- Лучше работать с двумя открытыми окнами (рис. 10.5) или с двумя открытыми вкладками (рис. 10.6) и перемещаться между ними, чем копировать, закрывать файл, открывать новый файл и вставлять в него скопированный код. Тогда будет проще сравнивать и перемещаться между окнами.
- Нужно сразу же после вставки проверять код в работе. Тогда можно будет быстро отловить потенциальные ошибки и отследить их, пока вы еще помните, какой код был только что вставлен.

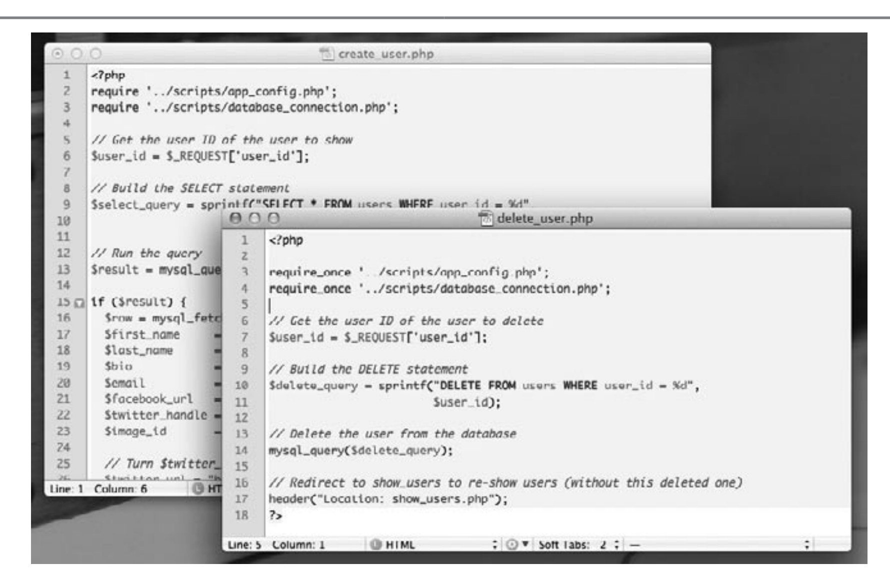


Рис. 10.5. При достаточно большом экране лучше всего при вырезке, копировании и вставке просматривать сразу два кодовых фрагмента, расположив их рядом

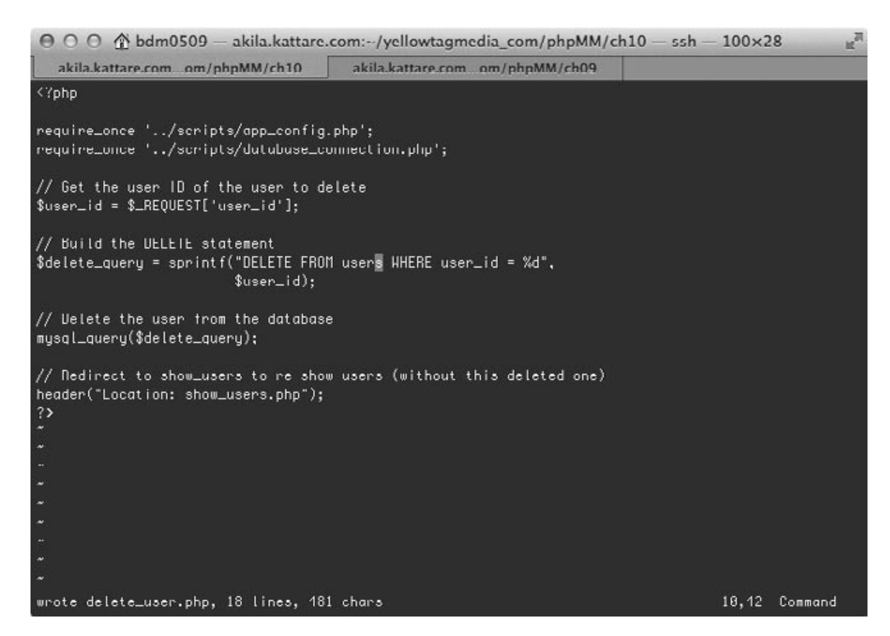


Рис. 10.6. При ограниченном пространстве экрана или если вы предпочитаете работать с более компактными формами, воспользуйтесь вкладками своего редактора (здесь показан Terminal компьютера Mac)

Вот и все! Не забывайте эти предостережения и не бойтесь заниматься вырезкой и вставкой. Это весьма важный инструмент в вашем арсенале.

Попробуйте все это в работе. У вас уже есть show_users.php с правильными ссылками, поэтому откройте окно этого сценария и выберите жертву для удаления. Щелкните на крестике, и вы получите результат, показанный на рис. 10.7. Перечень пользователей похож на тот, что был показан на рис. 10.3, но уже без пользователя по имени Питер Гэбриэл.



Рис. 10.7. Результат удаления пользователя

Удаление пользователей не должно быть некой тайной операцией

Теперь у вас есть превосходная возможность удаления пользователей. Нет никаких препятствий, пауз, ничего, кроме быстрого запроса delete_user.php, удаления информации из базы данных и возвращения к сценарию show_users.php.

И это совершенство с минимальной задержкой и ничем больше говорит только о том, что вам никогда не приходилось удалять пользователей.

Удаление является весьма важной операцией. Вы выбрасываете информацию, о которой уже никогда ничего не услышите. И это делается одним щелчком кнопкой мыши, без последующего предупреждения и возможности подумать еще раз. Проблема именно в этом.

Подумайте о своем собственном опыте работы в сети. Приходилось ли вам когда-нибудь что-нибудь удалять одним щелчком? Большинство из вас, наверное, сталкивались с целым потоком вопросов вроде: «Вы уверены?», «Вы уже никогда не сможете воспользоваться этим файлом» и даже «Осторожно! Ваша информация будет уничтожена безвозвратно!», оформленных в виде предупреждений. А ведь они являются важной частью процесса удаления.

Поэтому удаление нуждается в дополнении. Сейчас вам нужно заняться тем, что предшествует удалению. Пользователю нужно дать шанс еще раз обдумать свое решение, прежде чем управление будет передано сценарию delete_user.php. Поэтому вернемся к сценарию show_users.php.

Начнем с небольшого фрагмента на JavaScript

Когда вопрос касается таких вещей, как окна подтверждения, вы точно находитесь в мире браузеров и клиентов. Можно, конечно, создать какое-нибудь подтверждение на PHP, но это будет далеко не лучшим решением. Тогда вам понадобится отправить запрос об удалении серверу, тот должен будет запустить PHP-сценарий, создающий новую HTML-форму и запрашивающий подтверждение, браузер должен вернуть это пользователю, и пользователь должен щелкнуть на кнопке OK. Затем должен быть запущен еще один запрос, и тогда вы наконец-то доберетесь до удаления.

Даже если для сокращения количества обновлений страницы используется Ajax, то все равно для простого подтверждения осуществляется слишком много обращений к серверу. Особенно это становится понятно, если учесть что JavaScript предоставляет вам встроенные, доступные на всех клиентских машинах средства реализации точно такого же подтверждения.

Поэтому откройте еще раз сценарий show_users.php и добавьте в него немного кода JavaScript:

```
<?php

// Выбор всех пользователей с помощью инструкции SELECT
?>

<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
  <script type="text/javascript">
    function delete_user(user_id) {
      if (confirm("Вы уверены, что хотите удалить этого пользователя?" +
                  "\nВернуть его уже не удастся!")) {
        window.location = "delete_user.php?user_id=" + user_id;
      }
    }
  </script>
 </head>
 <body>
   <!-- код HTML -->
 </body>
</html>
```

В этом сценарии нет ничего сложного. Вы просто создаете функцию, запрашивающую у пользователя подтверждение перед тем, как передать управление сценарию delete_user.php. В сценарии выполняется и небольшое дополнительное

действие, связанное с тем, что идентификатор пользователя user_id должен быть передан этой функции, которая ставит его сразу же за delete_user.php в JavaScript-версии перенаправления под названием window.location.

ПРИМЕЧАНИЕ

Если для вас этот код немного непонятен, изучите дополнительный материал по JavaScript.

Если вы чувствуете себя неуверенно из-за того, что на этой странице используется не ссылка на внешний файл, а код JavaScript, прочитайте следующую врезку.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

А может быть, надо было воспользоваться внешней функцией?

Да, некоторых, наверное, напугал этот небольшой фрагмент кода. Но с его присутствием все же можно согласиться. После того как вы прочитали замечание по поводу вырезки и вставки, а затем увидели этот код, вы, наверное, уже готовы выбросить эту книгу в окно. (Или, может быть, вы обрадовались и собрались выбросить в окно какие-нибудь другие книги?)

Что тут может вызвать такое беспокойство? Посмотрим на следующий код:

```
<head>
  <link href="../css/phpMM.css"
        rel="stylesheet" type="text/css"
/>
  <script type="text/javascript">
    function delete_user(user_id) {
      // код подтверждения и перенаправления
    }
  </script>
 </head>
```

Почти так же часто, как и осуждение копирования и вставки, встречается и настоятельный совет никогда и ни при каких условиях не использовать JavaScript таким вот образом в head-блоке страницы. Подобная проблема рассматривается в большинстве книг примерно следующим образом.

1. Научитесь создавать небольшие JavaScript-фрагменты.
2. Научитесь создавать эффективные JavaScript-фрагменты.
3. После того как «наберетесь мастерства», научитесь помещать JavaScript во внешние файлы!
4. Научите всех своих друзей, приступающих к изучению JavaScript, тому же самому.

Звучит хорошо, и всем нравится эта небольшая догма. Но посмотрите на исходный код таких страниц, как www.amazon.com, www.google.com или www.apple.com. Каждый из этих веб-гигантов имеет <script>-теги с кодом внутри в head-области страницы!

> Разве писательский мир состоит из авторов, являющихся более квалифицированными, аккуратными и организованными программистами, чем высокооплачиваемые специалисты компаний Apple, Amazon и Google?
>
> Конечно, нет. Правда заключается в том, что есть масса случаев, когда страница нуждается в присутствии на ней JavaScript. И это вполне справедливо для страницы, над созданием которой вы работаете.
>
> Если у вас есть сервисные функции, например для создания универсальных диалоговых окон в библиотеке jQuery (которая настроена на такие вещи) или для проверки приемлемости конкретных типов данных, поместите их в файл сценария и ссылайтесь на них на всех ваших страницах. Это примерно то же самое, что вы делали для предназначенного для всего сайта CSS-файла, а также на сервере при работе с `app_config.php` и `database_connection.php`.
>
> Но только что написанная JavaScript-функция `delete_user` пригодится лишь для одной этой страницы! Она не относится к сервисному сценарию, предназначенному для всего сайта, и если ее туда добавить, она будет его только засорять. Можно, конечно, создать внешние сценарии для каждой страницы на своем сайте, но какая же это будет неразбериха!
>
> Иногда уместный фрагмент кода JavaScript в `head`-разделе страницы — это именно то, что нужно. Конечно, не следует засорять страницу многочисленными JavaScript-вставками между `p`-элементами и в промежутках между смежными `td`-элементами. Но не стоит бояться вставлять фрагменты JavaScript в свою страницу. Как и в случае с копированием и вставкой кода, к использованию этого приема нужно относиться разумно.

Внесение изменений в ссылки

После вставки кода JavaScript настало время для завершения задачи: изменения на странице ссылки, которая до этого вела непосредственно к сценарию `delete_user.php`, а теперь должна вести к вызову новой функции JavaScript:

```
<?php
  while ($user = mysql_fetch_array($result)) {
    $user_row = sprintf(
      "<li><a href='show_user.php?user_id=%d'>%s %s</a> " .
      "(<a href='mailto:%s'>%s</a>) " .
      "<a href='javascript:delete_user(%d);'><img " .
         "class='delete_user' src='../images/delete.png' " .
         "width='15' /></a></li>",
      $user['user_id'], $user['first_name'], $user['last_name'],
      $user['email'], $user['email'], $user['user_id']);
    echo $user_row;
  }
?>
```

Великолепно! Испытайте этот код в работе, и вы наконец-то получите показанное на рис. 10.8 полезное предупреждение перед тем, как запись, касающаяся пользователя по имени Уильям Шетнер, будет удалена.

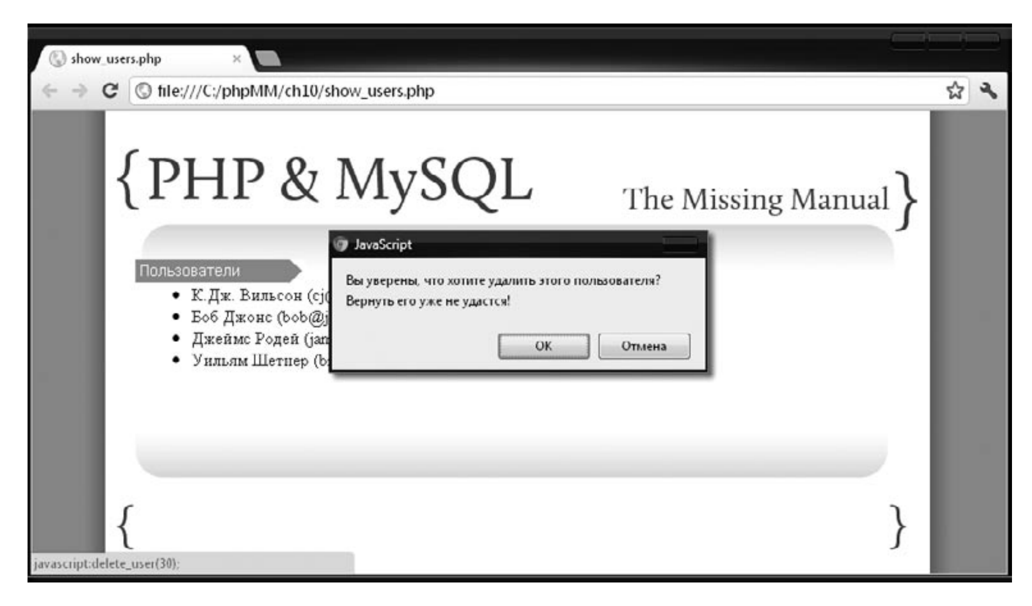


Рис. 10.8. Предупреждение перед удалением пользователя

Возражения, высказываемые вашим пользователям

Добавление диалогового окна предупреждения с подтверждением играет большую роль в клиентской части процесса удаления. Это окно дает пользователям шанс еще раз подумать об удалении пользователя и об отмене операции в случае неудовлетворенности этим решением или каких-то сомнений. Но это только половина уравнения. Нужно не только убедиться в намерениях осуществить удаление, но и в том, что это удаление состоялось.

Понятно, что вы как программист написали код, запустили этот код на выполнение и можете даже вернуться к базе данных и выдать свою собственную инструкцию `SELECT`, чтобы убедиться, что результатом вызова сценария `delete_user.php` действительно стало удаление. И что пользователь, разумеется, попал на страницу сценария `show_users.php`. Но все это с точки зрения программиста.

А для пользователя этого недостаточно. Он не только хочет получить подтверждение на удаление до того, как это удаление состоится. Он обычно также желает развеять все сомнения и узнать, что удаление состоялось. Значит, в конце процесса он хочет получить некое сообщение, подтверждающее то, что толь-

ко что случилось. Следовательно, ход сценария должен выглядеть примерно так.

1. Пользователь выбирает удаление другого пользователя щелчком на красном крестике, следующем за именем этого пользователя в окне сценария show_users.php.
2. Пользователь подтверждает свое намерение осуществить удаление.
3. Сценарий delete_user.php удаляет выбранного пользователя.
4. Пользователю предоставляется сообщение, говорящее примерно следующее: «Да, он от нас ушел, ушел, ушел».
5. Сценарий show_users.php снова показывает пользователей за вычетом того, который был удален.

Итак, новым здесь является шаг 4, над реализацией которого нужно немного поработать.

У перенаправления есть некоторые ограничения

Если посмотреть на ход выполнения сценария, может показаться, что естественным местом для отображения подтверждения является сценарий delete_user.php. Этот сценарий проводит удаление и выполняется перед выполнением сценария show_users.php, заново показывающего всех пользователей после вывода подтверждения об удалении.

Следовательно, как только удаление будет завершено, вы можете, например, выдать сообщение в строке состояния или вывести окно предупреждения. Но посмотрите на последнюю строку кода сценария delete_user.php:

```
header("Location: show_users.php");
```

Перенаправление в PHP осуществляется с помощью заголовков HTTP. Следовательно, эта строка отправляет браузеру обыкновенный заголовок Location. Браузер получает данный заголовок и отправляет в ответ HTTP-запрос по указанному URL. Все вроде бы просто, и все это неплохо работает.

Но, и это весьма важное «но», функция header может быть вызвана только до того, как из PHP будет отправлен какой-нибудь вывод. В файле не может использоваться инструкция echo, HTML, пустые строки или что-нибудь еще. Браузер может только получить заголовки, а затем передать запрос. Следовательно, на самом деле вы не можете что-либо отправить до вызова функции header, а как только будет вызвана функция header, вы не должны ничего отправлять после вызова. Разумеется, ошибки допускаются тогда, когда случается то, что не должно случаться, и поэтому каждый вызов заголовка Location сопровождается небольшой инструкцией exit(), чтобы гарантированно не было попыток выполнить что-нибудь еще.

Иными словами, такой сценарий, как delete_user.php, может работать с базой данных или другими объектами PHP, но не может осуществлять какой-либо вывод.

Он только лишь удаляет пользователя, а затем перенаправлет на сценарий просмотра, такой как show_users.php. Следовательно, вам нужно найти способ взаимодействия с show_users.php и позволить этому сценарию заняться оповещением пользователя об удалении.

ЗАМЕТКИ О ПРОЕКТИРОВАНИИ

«Модель-представление-контроллер» (или что-то вроде этого)

Мы приступаем к рассмотрению важной схемы веб-приложения. Она называется MVC, что означает Model-View-Controller («Модель-представление-контроллер»). По этой схеме у вас есть три категории операций: модели, представления и контроллеры. Если строго придерживаться схемы MVC, данные три категории никогда не перекрывают друг друга.

Сначала идет модель, которая взаимодействует с базой данных. Модель олицетворяет, или моделирует, информацию вашего приложения. В вашем приложении сценарий delete_user.php использует MySQL напрямую. При более формальном MVC-подходе у вас были бы PHP-объекты, такие как User.php, с такими методами, как delete() или remove(). И вы могли бы создавать примерно следующий код:

```
User user_to_delete =
  User.find_by_id($user_id);
user_to_delete.delete();
```

Можно прийти к выводу, что модельная часть MVC — это то, что взаимодействует с базой данных. Что касается вашего кода, то у вас нет четкой модели, но при этом несомненно совершаются многочисленные операции взаимодействия с базой данных.

Затем идет представление, которое показывает информацию пользователю. В вашем приложении такие сценарии show_user.php и show_users.php являются в некоторой степени представлениями. Они заполнены HTML и информацией. Причина, позволяющая считать их представлениями только «в некоторой степени», заключается в том, что некоторые их действия можно отнести к поведению контроллера.

И третьей категорией в MVC-архитектуре являются контроллеры. Контроллер управляет потоком информации. Он использует модель для получения информации от базы данных или из хранилища данных, а затем передает эту информацию классам представления или сценариям, которые ее отображают. Очень похож на контроллер сценарий delete_user.php. Даже притом, что он не использует модель, а обращается к базе данных непосредственным образом, он осуществляет определенные действия, а затем передает управление представлению, в качестве которого выступает сценарий show_users.php.

В большинстве веб-приложений, написанных на PHP, у вас не будет строгой MVC-структуры. Чтобы, работая с PHP, создать полноценную MVC-структуру, нужно приложить массу усилий. Обычно получается более смешанный подход, при котором такие сценарии, как delete_user.php (которые можно отнести к контроллерам), передают информацию сценариям типа show_users.php (представлениям). Но в сценарии

delete_user.php имеются также аспекты модели (относительно непосредственного обращения к базе данных). А в сценарии show_users.php присутствуют аспекты контроллера и модели, поскольку в нем определяется, что именно показывать, и он извлекает информацию непосредственно из базы данных.

Если вы не можете реализовать на PHP MVC-архитектуру, то к чему все это длинное повествование? На это есть две веские причины. Во-первых, вам постоянно будут попадаться сведения о MVC, и если вы сможете связать свою работу по созданию веб-приложений с MVC и с тем, чем могут заниматься ваши приятели, то вы сможете существенно повысить свою популярность. И во-вторых, если вы сможете четко определить, чем занимаются ваши сценарии, то зачастую вам значительно быстрее удастся определить и то, как это будет делаться.

Что касается сценария delete_user.php, вы видите, что в основном он выполняет функцию контроллера. Поэтому есть вполне определенный смысл передать некую информацию тому сценарию, который главным образом является представлением, то есть сценарию show_users.php, и позволить ему управлять отображением информации пользователю.

Следовательно, от сценария delete_user.php требуется обеспечить сообщение, поскольку он знает, что удаление состоялось, но показывать его этот сценарий должен поручить какому-нибудь другому сценарию. Сообщение можно добавить к вашему перенаправлению. Подключите это новое сообщение к новому параметру запроса, который называется success_message:

```
// Перенаправление на show_users, чтобы снова показать пользователей
// (без удаленного пользователя)
$msg = "Указанный пользователь был удален.";
header("Location: show_users.php?success_message={$msg}");
```

СОВЕТ

Если вы подумали также, что было бы неплохо иметь и сообщение об ошибке error_message, то вы на абсолютно правильном пути.

Теперь, еще до того, как вернуться к работе над кодом представления в сценарии show_users.php, вы можете все это протестировать. Зайдите на страницу сценария show_users.php, удалите пользователя, а затем внимательно посмотрите на адресную строку браузера, когда будете возвращены обратно на страницу сценария show_users.php. Вы должны увидеть параметр запроса success_message со значением, в качестве которого фигурирует ваше сообщение (рис. 10.9).

В самом окне смотреть не на что. Однако вы можете увидеть, что сообщение, которое сценарий delete_user.php добавил к URL, отправленному браузеру, содержит полезное значение: тот самый текст, который вы хотели бы видеть в красивом окне предупреждения или в строке состояния. Вот и хорошо: теперь у вас есть свой код представления, занимающийся показом этого сообщения пользователю.



Рис. 10.9. Сообщение в адресной строке браузера

Возвращение окна предупреждения, создаваемого с помощью JavaScript

Теперь нужно вернуться к сценарию show_users.php, и вы получите входящее сообщение.

ПРИМЕЧАНИЕ

У вас действительно есть потенциальное входящее сообщение. Когда сценарий show_users.php вызывается обычным образом, у него нет сообщения. Оно появляется только при перенаправлении после удаления (или какой-нибудь подобной операции) благодаря параметрам запроса.

Что должно произойти при получении сообщения? Наверное, проще всего вернуться к JavaScript и воспользоваться диалоговым окном предупреждения. Это то же самое, что и диалоговое окно подтверждения, применяемое перед удалением, и оба эти окна составляют неплохую взаимодополняющую пару.

Подход, в котором целиком используется JavaScript

Одним из подходов будет создание функции на JavaScript, которую можно добавить к сценарию show_users.php. JavaScript не имеет непосредственной поддержки чтения параметров запроса, поэтому для их получения вам придется провести небольшой разбор. Для этого понадобится что-то, использующее регулярные выражения для извлечения части значения свойства window.location.href, которое имеется у URL браузера:

```
function get_request_param_value(param_name) {
  param_name = param_name.replace(/[\[]/,"\\\[").replace(/[\]]/,"\\\]");
  var regexS = "[\\?&]" + param_name + "=([^&#]*)";
  var regex = new RegExp(regexS);
```

```
  var results = regex.exec(unescape(window.location.href));
  if (results == null)
    return "";
  else
    return results[1];
}
```

ВНИМАНИЕ

Вам не нужно вникать в этот код. Но если вы захотите потратить несколько минут на построчный разбор этого кода, то вы существенно повысите свой уровень владения JavaScript. В коде еще раз демонстрируется, что несмотря на то, что на первый взгляд регулярные выражения могут выглядеть слишком странно, они являются очень важной частью инструментария программиста. Подумайте также о том, что каждая крупица изученного о регулярных выражениях в этой книге, посвященной PHP, переносится и на JavaScript.

Затем вы можете вызвать эту функцию показанным здесь образом, чтобы добраться до параметра `success_message` (возможно, в другой JavaScript-функции):

```
msg = get_request_param_value("success_message");
if (msg.length > 0) {
  // уведомление в адрес пользователя
}
```

Итак, после того как у вас прошло косоглазие от всех этих прямых и обратных слэшей, имеющихся в показанном ранее коде функции `get_request_param_value`, вы сможете выдать сообщение в окне предупреждения:

```
msg = get_request_param_value("success_message")
if (msg.length > 0) {
  alert(msg);
}
```

Этот подход не имеет никаких изъянов. Он превосходно работает, и если добавить этот код в `head`-раздел сценария `show_users.php` между `script`-тегами, вы получите примерно такое же сообщение, как показано на рис. 10.10.

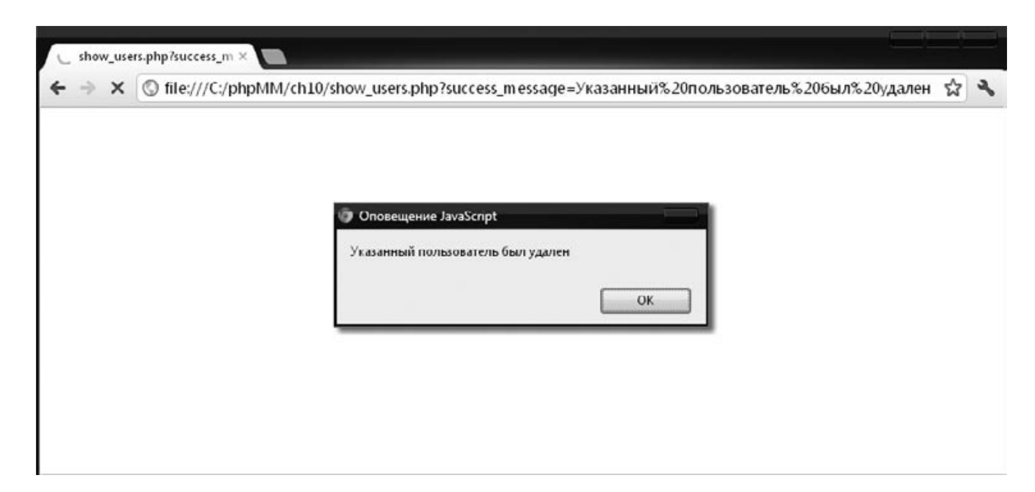


Рис. 10.10. Сообщение об удалении пользователя

ПРИМЕЧАНИЕ

Показанный на рис. 10.10 серый фон — это артефакт, присущий вызову функции alert. Возможно, вам захочется еще больше улучшить пользовательское восприятие, не запуская функцию alert до тех пор, пока не будет загружен документ. Можно воспользоваться свойством window.onload, событием onload, относящимся к тегу body, или различными имеющимися в библиотеке jQuery способами запуска кода после загрузки документа, добившись тем самым более качественного восприятия.

До того как вы начнете задаваться вопросом о сборе всего этого в единое целое, следует сказать, что существует более подходящий способ решения этой задачи.

Ваш PHP управляет выводом

В показанном выше подходе, полностью базирующемся на применении JavaScript, сделано тонкое, но весьма важное предположение: решение о том, что делать, что показывать и как действовать, должно приниматься на странице, то есть в HTML, CSS и JavaScript, которые доставлены пользователю через его браузер. Именно JavaScript должен определить, был ли передан параметр success_message. Он же должен разобрать URL запроса и найти значение этого параметра, и он же должен при определенных условиях показать сообщение в окне предупреждения.

Но дело в том, что сценарий show_users.php не имеет таких же ограничений, накладывающихся на страницу, которую он выводит. Только то, что HTML и JavaScript, которые он в конечном счете выводит, ничего не знают о том, был или нет параметр запроса, еще не означает, что ваш сценарий, генерирующий этот вывод, также находится в неведении на этот счет. И вы можете просто добавить параметр запроса в сценарий show_users.php, чем вы уже неоднократно занимались:

```
$msg = $_REQUEST['success_message'];
```

И теперь одной строкой вы убираете весь этот JavaScript:

```
function get_request_param_value(param_name) {
  param_name = param_name.replace(/[\[]/,"\\\[").replace(/[\]]/,"\\\]");
  var regexS = "[\\?&]" + param_name + "=([^&#]*)";
  var regex = new RegExp(regexS);
  var results = regex.exec(unescape(window.location.href));
  if (results == null)
    return "";
  else
    return results[1];
}
```

По любым меркам это победа.

ПРИМЕЧАНИЕ

Возможно, неплохо бы было еще добавить такую функцию, как get_request_param_value к вашим основным сервисным функциям JavaScript и пользоваться ею при отсутствии кода PHP, который генерирует вывод.

Но у вас есть веский повод ухватиться за этот вариант: вы управляете всем, что отправляется клиенту. Ваш сценарий может принимать решения о том, что ему нужно выводить.

Следовательно, в своем коде PHP вы можете сделать нечто подобное:

```
// Проверка наличия сообщения, предназначенного для отображения
if (isset($_REQUEST['success_message'])) {
  $msg = $_REQUEST['success_message'];
}
```

Это делается на сервере. Пока вы еще ничего не выводили. Если имеется сообщение для отображения (и только в этом случае), можно просто добавить несколько строк JavaScript в свой HTML-вывод:

```
  <script type="text/javascript">
    function delete_user(user_id) {
      if (confirm("Вы уверены, что хотите удалить этого пользователя?" +
                  "\nВернуть его уже не удастся!")) {
        window.location = "delete_user.php?user_id=" + user_id;
      }    }

<?php if (isset($msg)) { ?>
    window.onload = function() {
      alert("<?php echo $msg ?>");
    }
<?php } ?>
  </script>
```

Итак, соберите все это вместе и получите новый, улучшенный сценарий show_users.php:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';

// Создание инструкции SELECT
$select_users =
  "SELECT user_id, first_name, last_name, email " .
  "  FROM users";

// Запуск запроса
$result = mysql_query($select_users);

// Проверка наличия сообщения, предназначенного для отображения
if (isset($_REQUEST['success_message'])) {
  $msg = $_REQUEST['success_message'];
}
?>

<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
  <script type="text/javascript">
    function delete_user(user_id) {
      if (confirm("Вы уверены, что хотите удалить этого пользователя?" +
                  "\nВернуть его уже не удастся!")) {
```

```
        window.location = "delete_user.php?user_id=" + user_id;
      }
    }

<?php if (isset($msg)) { ?>
    window.onload = function() {
      alert("<?php echo $msg ?>");
    }
<?php } ?>
  </script>
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пользователи</div>
  <div id="content">
   <ul>
     <?php
       while ($user = mysql_fetch_array($result)) {
         $user_row = sprintf(
           "<li><a href='show_user.php?user_id=%d'>%s %s</a> " .
           "(<a href='mailto:%s'>%s</a>) " .
           "<a href='javascript:delete_user(%d);'><img " .
              "class='delete_user' src='../images/delete.png' " .
              "width='15' /></a></li>",
           $user['user_id'], $user['first_name'], $user['last_name'],
           $user['email'], $user['email'], $user['user_id']);
         echo $user_row;
       }
     ?>
   </ul>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

ПРИМЕЧАНИЕ

Вполне возможно, что на данный момент будет трудно отследить все изменения в show_user.php и show_users.php, а также в app_config.php. Если обнаружится, что вы допустили какие-нибудь странные ошибки или получили необычные результаты, можно будет обратиться к веб-странице, посвященной серии недостающих руководств и загрузить примеры, встречающиеся в последних главах. Вы получите правильный, обновленный набор файлов и сможете сконцентрироваться на новых изменениях, а не на отладке старого кода.

Все, что здесь сделано, действительно играет важную роль в программировании на PHP. Вместо зависимости в вопросах поиска сложных решений от вашего вывода большинство решений принимается в PHP, а затем подстраивается в качестве результата к выводу. Таким образом, один сценарий может выдать два, три, четыре и даже больше вариантов одного и того же вывода.

Сначала нужно применить это к управлению тестами. Если вы по-прежнему получаете в адресной строке браузера URL, похожий на yellowtagmedia.com/phpMM/ch10/show_users.php?success_message=Указанный%20пользователь%20был%20удален,

то просто перезагрузите страницу для получения новых изменений, внесенных в show_users.php. Вы увидите красивое окно с сообщением, переданным через URL (рис. 10.11).



Рис. 10.11. Этот вывод выглядит вполне привлекательно

Посмотрите на исходный код страницы, чтобы понять, что в ней можно найти интересного. На рис. 10.12 показано, что в нем имеется «жестко запрограммированное» окно предупреждения с переданным сообщением. Измените текст сообщения в URL запроса, и вы увидите, что HTML изменился в соответствии с этим текстом.

show_users.php?success_m

view-source:file:///C:/phpMM/ch10/show_users.php?success_message=Указанный%20пользователь%20был%20удален

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
  <script type="text/javascript">
    function delete_user(user_id) {
      if (confirm("Вы уверены, что хотите удалить этого пользователя?" +
                  "\nВернуть его уже не удастся!")) {
        window.location = "delete_user.php?user_id=" + user_id;
      }
    }

    window.onload = function() {
      alert("Указанный пользователь был удален");
    }
  </script>
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: недостающее руководство</h1></div>
  <div id="example">Пользователи</div>

  <div id="content">
   <ul>
    <li><a href="show_user.php?user_id=1">К.Дж. Вильсон</a>(<a href="mailto:cj@texasrangers.com">cj@texasrangers.com</a>)
<a href='javascript:delete_user(1);'><img class="delete_user" src="../images/delete.png" width="15" /></a></li><li><a
href="show_user.php?user_id=22">Джеймс Родей</a>(<a href="mailto:james@roday.net">james@roday.net</a>)<a
href="delete_user.php?user_id=22"><img class="delete_user" src="../images/delete.png" width="15" /></a></li><li><a
href="show_user.php?user_id=30">Уильям Шетнер</a>(<a href="mailto:bill@williamshatner.com">bill@williamshatner.com</a>)<a
href="delete_user.php?user_id=30"><img class="delete_user" src="../images/delete.png" width="15" /></a></li>  </ul>
  </div>
  <div id="footer"></div>
```

Рис. 10.12. Исходный код страницы

А теперь удалите все параметры запроса из `show_users.php` в строке URL-адреса и еще раз вызовите страницу. Окно предупреждения пропадет, а вместе с ним пропадет и код JavaScript в HTML-странице, который генерируется сценарием `show_users.php`. На рис. 10.13 показан исходный код этой страницы: функции `window.onload` в нем уже нет.

```
show_users.php
view-source:file:///C:/phpMM/ch10/show_users.php

<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
  <script type="text/javascript">
    function delete_user(user_id) {
      if (confirm("Вы уверены, что хотите удалить этого пользователя?" +
                  "\nВернуть его уже не удастся!")) {
        window.location = "delete_user.php?user_id=" + user_id;
      }
    }

  </script>
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: недостающее руководство</h1></div>
  <div id="example">Пользователи</div>

  <div id="content">
   <ul>
    <li><a href="show_user.php?user_id=1">К.Дж. Вильсон</a>(<a href="mailto:cj@texasrangers.com">cj@texasrangers.com</a>)
<a href='javascript:delete_user(1);'><img class="delete_user" src="../images/delete.png" width="15" /></a></li><li><a
href="show_user.php?user_id=22">Джеймс Родей</a>(<a href="mailto:james@roday.net">james@roday.net</a>)<a
href="delete_user.php?user_id=22"><img class="delete_user" src="../images/delete.png" width="15" /></a></li><li><a
href="show_user.php?user_id=30">Уильям Шетнер</a>(<a href="mailto:bill@williamshatner.com">bill@williamshatner.com</a>)<a
href="delete_user.php?user_id=30"><img class="delete_user" src="../images/delete.png" width="15" /></a></li>  </ul>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

Рис. 10.13. Измененный код страницы

При таком подходе стоит задуматься вот о чем: как работают закладки? Поскольку параметры запроса являются частью URL (или в данном случае не являются частью URL), механизм создания закладок присоединит к списку имеющихся в закладках URL с конкретным вариантом сообщения. Это означает, что вам придется разбираться с тем, что произойдет, если кто-нибудь поместит в закладки данную страницу с отображаемым сообщением. Данное сообщение или, если быть точнее, одно из этих сообщений будет показываться при каждом вызове этой ссылки.

Функция alert прерывает действия

Теперь у вас есть привлекательное предупреждение, состоящее из двух частей. Окно подтверждения получает пользовательское одобрение в виде `OK` перед удалением пользователя, а другое окно предупреждения оповещает пользователей о том, что удаление состоялось. С функциональной точки зрения вы готовы к продолжению работы.

Но сейчас наступил один из тех моментов, когда вам нужно отвлечься от веб-программирования и приступить к обдумыванию веб-дизайна или, точнее, вопросов удобства пользования вашими веб-элементами. Удобство пользования — это просто краткая форма следующего вопроса: «Каким будет пользовательское восприятие всего этого?»

ПРИМЕЧАНИЕ

Часто в разговорах на подобные темы можно услышать такие термины, как UX (что означает user experience, пользовательское восприятие) и UI (user interface, пользовательский интерфейс). В некоторой степени эти термины не столь далеки друг от друга, хотя проектировщик UX может быть недоволен, если его станут путать с проектировщиком UI. Но основная цель их работы одна и та же: создание естественного и интересного интерактивного пользовательского восприятия. Это восприятие включает не только функциональные возможности, но и такие понятия, как эстетика, доступность и общее «ощущение» веб-сайта и веб-приложения.

Что касается удаления пользователя, здесь все выглядит весьма конкретно. Хотя вы можете применить библиотеку jQuery, чтобы предоставить пользователю более приглядные диалоговые окна, все же есть вполне определенный смысл в том, чтобы прервать на время действия пользователя, чтобы он убедился в своем намерении удалить пользователя. В результате вам требуется двойное действие: щелчок при выборе удаляемого пользователя и еще один щелчок, чтобы удостовериться в серьезности намерений.

ПРИМЕЧАНИЕ

Если вам больше нравится привлекательное диалоговое окно и окно подтверждения в стиле jQuery, то в таком случае можете изучить пользовательский интерфейс jQuery и имеющиеся в этой библиотеке диалоговые окна на веб-сайте www.jqueryui.com/demos/dialog. В частности, нужно посмотреть настройки Modal-подтверждения. Загрузка и установка jQuery UI займет 10 минут, и еще 5 минут уйдет на переход от вашего вызова подтверждения к вызову диалогового окна подтверждения, имеющегося в jQuery.

А что же после удаления? Разумеется, вам следует уведомить пользователей, о том, что удаление состоялось. Но нужно ли тормозить всю их работу до тех пор, пока они не щелкнут на кнопке `OK`? В идеале им необходимо сообщить об удалении, но сделать это, не прерывая их работу.

В удобстве пользования веб-приложением, в дизайне, или как вы это еще назовете, есть важный принцип. Если вы собираетесь заставить пользователя убрать руки с клавиатуры и щелкнуть на кнопке, нужно иметь для этого веские основания. В данном случае вы рискуете вызвать у пользователя раздражение: «Зачем щелкать еще раз? Я же только что дважды щелкнул для удаления пользователя!»

Приведение сообщений к единому стандарту

Есть еще один вопрос, который, возможно, уже возник у вас: неужели сообщение об успешном удалении является единственным типом сообщения, нуждающимся в отображении? Что если у вас будет ошибка, не требующая вызова функции `handle_error`? Что если вам понадобится сообщение о каком-нибудь состоянии, что-нибудь вроде следующего: «Пожалуйста, зарегистрируйтесь, прежде чем предпринимать попытку удаления пользователя».

ПРИМЕЧАНИЕ

Регистрироваться перед удалением пользователя? Звучит неплохо и даже может стать темой для главы 11, как вы считаете? Будем ждать с нетерпением.

Все эти случаи одного порядка: вам нужно что-нибудь сообщить пользователям, но при этом не прерывать их работу. Можно добавить содержимое сообщения к содержимому страницы, но окна предупреждения и подтверждения, предлагаемые JavaScript, явно будут при этом не лучшими вариантами.

И при дальнейшем рассмотрении в идеале способ вывода сообщения должен носить универсальный характер. Не хочется, чтобы каждый сценарий выводил 5 или 10 страниц кода. Лучше иметь какой-нибудь вывод, похожий на этот:

```
<body>
  <?php display_messages($_REQUEST); ?>

  <!-- Весь остальной, необходимый вам HTML-вывод -->
</body>
```

А затем эта функция просто займется всем остальным, независимо от конечного смысла. Так, для сообщения об успешном завершении вы могли бы получить сообщение типа баннера, появляющегося в верхней части страницы (рис. 10.14).

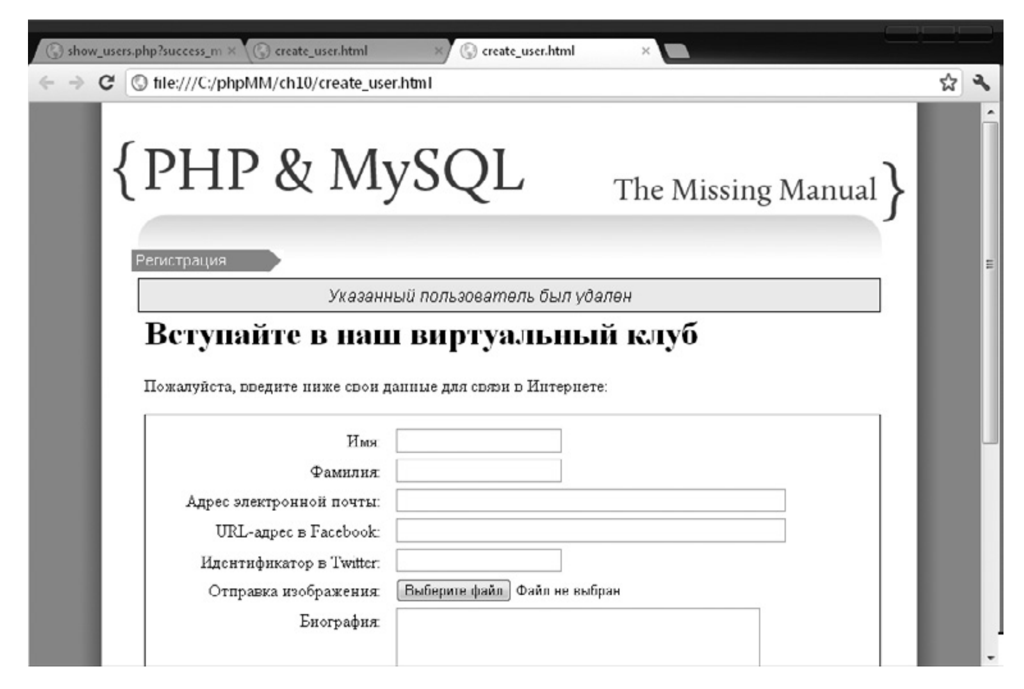


Рис. 10.14. Ненавязчивое сообщение доносит до пользователя сведения, не заставляя его щелкать на чем-нибудь или что-нибудь подтверждать

Код HTML для сообщения об успешном завершении имеет довольно простой вид:

```
<div id="messages">
  <div class="success">
```

```
    <p>Указанный пользователь был удален</p>
  </div>
</div>
```

Сообщения об ошибках могут быть показаны таким же способом, что изображен на рис. 10.15.

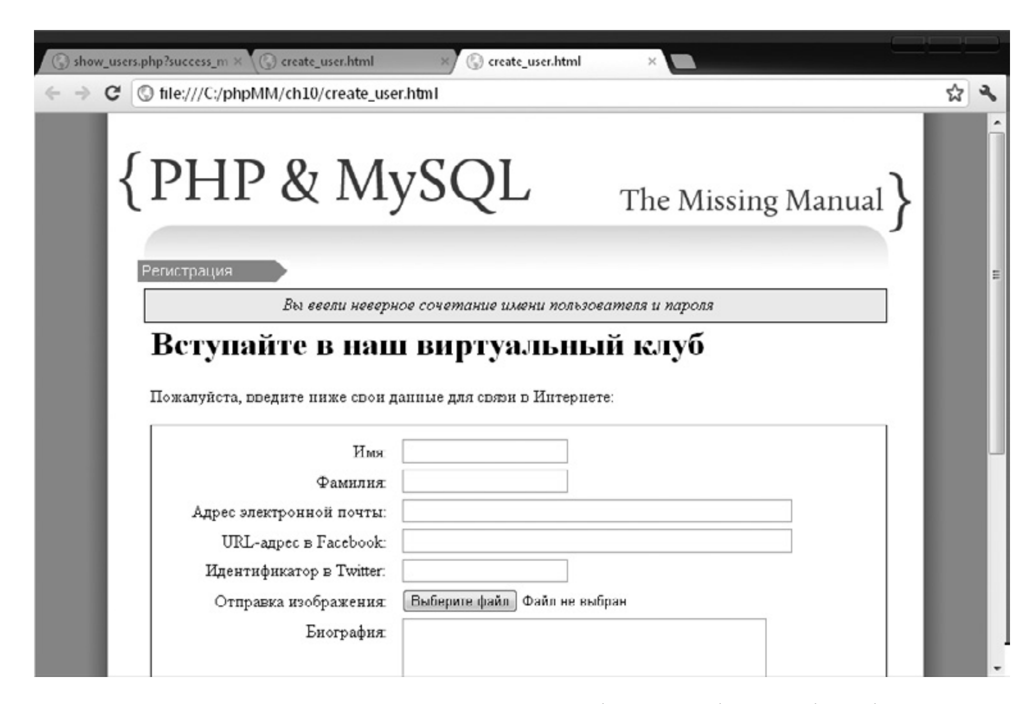


Рис. 10.15. Теперь у вас есть несколько способов для сообщения об ошибках в зависимости от серьезности конкретной ошибки

ПРИМЕЧАНИЕ

Возможно, вы уже заметили, что этот примитивный макет был сделан со сценарием create_user.html. Это просто был наиболее доступный фрагмент HTML на момент, когда нужно было посмотреть, как выглядят такие сообщения. Какая именно страница используется для данной проверки, не имеет значения. Следует помнить, что цель состоит в том, чтобы заставить любую страницу автоматически показывать или не показывать посланное ей сообщение.

А вот как выглядит код HTML для сообщения об ошибке. Он идентичен коду для сообщения об успешном завершении, но в нем используется другое значение атрибута `class` во внутреннем `div`-контейнере:

```
<div id="messages">
  <div class="error">
    <p>Вы ввели неверное сочетание имени пользователя и пароля</p>
  </div>
</div>
```

Создание новой сервисной функции для отображения

И все же вернемся к универсальному мышлению. Вместо того чтобы переживать, как конкретное сообщение об успешном завершении передается от delete_user.php к show_users.php, подумаем, как выглядит наиболее общая форма сообщения об успешном завершении.

Наверное, как-нибудь так:

```
<div id="messages">
  <div class="success">
    <p>$msg</p>
  </div>
</div>
```

Но это, конечно же, не настоящий код PHP. На самом деле нужен следующий код:

```
<div id="messages">
  <div class="success">
    <p><?php echo $msg; ?></p>
  </div>
</div>
```

Но в этом для вас вообще нет ничего сложного. Нужна всего лишь новая функция, которой передается сообщение:

```
function display_success_message($msg) {
  echo "<div id='messages'>\n";
  echo " <div class='success'>\n";
  echo "  <p>{$msg}</p>\n";
  echo " </div>\n";
  echo "</div>\n\n";
}
```

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

А как насчет функции sprintf? И зачем здесь такие элементы, как \n?

Наверное, способов создания функций вроде display_success_message не меньше, чем букв в алфавите. Для вставки сообщения можно воспользоваться и функцией sprintf. Можно собрать несколько вызовов инструкции echo в одну строку (применяя echo или sprintf). Можно вывести чистый HTML, прерывая его кодом PHP с использованием конструкции <?php и ?>. И в каждом случае ваше решение будет вполне приемлемым.

Интерес также вызывают сочетания символов \n. Они предназначены для того, чтобы просматриваемый исходный код был более понятен. Без них вывод смотрелся бы примерно так:

```
<div id='messages'> <div class='success'>
<p>{$msg}</p> </div></div>
```

То есть это была бы одна большая строка кода HTML. При задействовании символов перевода строки картина для пользователей не меняется. HTML не обрабатывает эти переводы строк. Но при просмотре исходного кода вы увидите более наглядный HTML:

```
<div id='messages'>
   <div class='success'>
      <p>{$msg}</p>
   </div>
</div>
```

Итак, нужны ли символы `\n`? Вообще-то нет. Помогают ли они пользователю? Нет. Но они определенно упрощают отладку и улучшают читаемость кода. Применять ли их? И нужны ли они при использовании `echo`, `sprintf` или обеих этих функций?

В вашем PHP-путешествии вы уже дошли до момента, когда все меньше категорий «правильно-неправильно» и все больше таких понятий, как «стиль» и «персональные предпочтения». Функцию `sprintf` можно применять везде для запросов и вывода и в любом месте между ними. Вы можете использовать `echo` для вывода и `sprintf` для запросов. Или, скорее всего, вы будете задействовать то, что вам заблагорассудится при создании того или иного сценария.

То же самое можно сказать о символах `\n` и о переводах строк. Иногда вы будете работать с полной самоотдачей, чтобы получить HTML-вывод красивым, четким и легко читаемым. А иногда вы будете понимать, что не сможете тратить несколько часов, пытаясь улучшить внешний вид для тех редких людей, которые вознамерятся просматривать исходный код. (И опять-таки можно сказать, что вы и есть тот самый редкий человек, оправдывающий прикладываемые для этого усилия.)

Итак, у вас есть работоспособная функция. А как насчет сообщений об ошибках? Можно воспользоваться примерно таким кодом:

```
function display_error_message($msg) {
  echo "<div id='messages'>\n";
  echo " <div class=error>\n";
  echo "  <p>{$msg}</p>\n";
  echo " </div>\n";
  echo "</div>\n\n";
}
```

Однако обе эти функции выводят `div`-контейнер `messages`. В этом нет ничего хорошего. Вам нужно что-то, что сможет обрабатывать оба типа сообщений. Затем такая родительская функция может передавать отдельные сообщения менее объемным функциям, каждая из которых обрабатывает удачные завершения и ошибки:

```
function display_messages($success_msg, $error_msg) {
  echo "<div id='messages'>\n";
  display_success_message($success_msg);
```

```
  display_error_message($error_msg);
  echo "</div>\n\n";
}

function display_success_message($msg) {
  echo " <div class='success'>\n";
  echo "   <p>{$msg}</p>\n";
  echo " </div>\n";
}

function display_error_message($msg) {
  echo " <div class='error'>\n";
  echo "   <p>{$msg}</p>\n";
  echo " </div>\n";
}
```

Вроде бы выглядит получше. Но… вас здесь ничего не смущает? Не замечаете ли вы некую продублированность кода?

Появление дубликатов — вполне ожидаемая проблема

Здесь эта проблема едва заметна, именно поэтому она может вызывать такое сильное раздражение. Посмотрите, как похожи друг на друга следующие две функции:

```
function display_success_message($msg) {
  echo " <div class='success'>\n";
  echo "   <p>{$msg}</p>\n";
  echo " </div>\n";
}

function display_error_message($msg) {
  echo " <div class='error'>\n";
  echo "   <p>{$msg}</p>\n";
  echo " </div>\n";
}
```

В них очень много одинакового кода. И все, что их отличает друг от друга, — значение атрибута `class` в `div`-контейнере. При виде такого похожего кода у вас должна возникать мысль о его слабости, которой нужно избегать.

Повторяющийся код позволяет объединить эти функции:

```
function display_message($msg, $msg_type) {
  echo " <div class='{#msg_type}'>\n";
  echo "   <p>{$msg}</p>\n";
  echo " </div>\n";
}
```

Данный код выглядит значительно лучше. Он понятен, краток и, как объясняется в следующей врезке, отлично вписывается в DRY-концепцию. Пойдем еще дальше и определим типы разрешенных сообщений в виде констант:

```
define("SUCCESS_MESSAGE", "success");
define("ERROR_MESSAGE", "error");

function display_messages($success_msg, $error_msg) {
  echo "<div id='messages'>\n";
  display_message($success_msg, SUCCESS_MESSAGE);
  display_message($error_msg, ERROR_MESSAGE);
  echo "</div>\n\n";
}

function display_message($msg, $msg_type) {
  echo " <div class='{#msg_type}'>\n";
  echo "  <p>{$msg}</p>\n";
  echo " </div>\n";
}
```

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Создание DRY-кода

Продолжая заниматься программированием, вы услышите о том, как люди начинают говорить о DRY-коде или о том, что код нужно вычищать (Drying up your code, чисти свой код). В обоих этих выражениях в качестве акронима используется слово DRY, которое означает Don't Repeat Yourself, то есть: «Не нужно повторяться». С этой точки зрения вы проделали хорошую работу. Помните главу 4, где вы перемещали некоторые основные константы, использующиеся во всем приложении, в файл `app_config.php`? Тем самым гарантировалась неповторяемость таких констант (или ваших действий) сразу в нескольких файлах. Вы помещали их в одно место, а затем все другие сценарии ссылались на это место.

То же самое можно сказать и о создании сценария `database_connection.php`. Здесь также вместо многократного повторения кода подключения вы убрали этот код из многочисленных мест и поместили его в одно место. Тем самым вы очистили код с применением DRY-подхода: превратили его в DRY-код и удалили весь повторяющийся код отовсюду, где это было возможно.

Что же касается функций `display_success_message` и `display_error_message`, их преобразование носило менее масштабный характер. Речь шла всего лишь о трех строчках кода, не так ли? И все же если вы можете записать эти три строчки кода в одном месте и сослаться на них из двух мест, вы повысите качество своего проекта. Вы позаботились о том, что при необходимости изменить порядок вывода сообщений нужно будет исследовать код в одном месте, а не в двух. Это хороший стиль программирования, в результате которого получается DRY-код и все ваши коллеги подумают о том, что это работа мастера.

Великолепно! Теперь не нужно запоминать, каким был тип сообщения для ошибки: ERROR, error, errors или вообще каким-нибудь другим. Отображением в данном случае займется константа.

Теперь можно приступать к сбору кода в единое целое. Создайте новый сценарий и назовите его view.php. Затем поместите в него весь этот код наряду с инструкцией require_once для включения обязательного сценария app_config.php:

```
<?php

require_once 'app_config.php';

define("SUCCESS_MESSAGE", "success");
define("ERROR_MESSAGE", "error");

function display_messages($success_msg, $error_msg) {
  echo "<div id='messages'>\n";
  display_message($success_msg, SUCCESS_MESSAGE);
  display_message($error_msg, ERROR_MESSAGE);
  echo "</div>\n\n";
}

function display_message($msg, $msg_type) {
  echo " <div class='{$msg_type}'>\n";
  echo "  <p>{$msg}</p>\n";
  echo " </div>\n";
}

?>
```

ПРИМЕЧАНИЕ

Пока ничего из сценария app_config.php в сценарии view.php не используется. И тем не менее, поскольку этот сценарий является местом, в котором находится вся ваша основная информация, вполне вероятно, что рано или поздно он вам понадобится. Наверное, лучше запросить его с помощью require_once сейчас, чтобы он был доступен.

Коды сценариев View и Display имеют общий характер

Теперь у вас есть еще один сценарий — view.php. Он находится в вашем основном каталоге scripts/ наряду с такими сценариями, как app_config.php и database_connection.php. Это также способствует тому, чтобы вы создавали не только полезный, но и хорошо организованный код. И хотя вы можете поместить функции display_messages и display_message в сценарий app_config.php, примером хорошей организации это не станет.

Теперь следует потратить время на создание групп функций в сценариях, которые к тому же имели бы понятные названия, и это время будет потрачено не зря. При создании таких сценариев, как show_users.php, которые занимаются отображением данных, вы сразу же понимаете, что можете включить в них сценарий view.php и получить в свое распоряжение полезные функции. С другой стороны, в таких

сценариях, как delete_user.php, которые не занимаются отображением данных, view.php можно не использовать. Это элементарно.

ПРИМЕЧАНИЕ

Разумеется, точно такой же принцип касается и сценария database_connection.php. Если подключаться к базе данных не нужно, то и инструкция require_once database_connection.php будет не нужна. Если чем-то заниматься, то это нужно делать всерьез. Все значительно упрощается, когда сценарии хорошо организованы и названы в соответствии с их функциями.

Интеграция утилит, представлений и сообщений

И наконец, вы готовы собрать все вместе. Еще раз просмотрите сценарий show_users.php и менее совершенную систему сообщений, с которой началось все это путешествие, приведшее к созданию сценария view.php:

```
<head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
  <script type="text/javascript">
    function delete_user(user_id) {
      if (confirm("Вы уверены, что хотите удалить этого пользователя?" +
                  "\nВернуть его уже не удастся!")) {
        window.location = "delete_user.php?user_id=" + user_id;
      }
    }

<?php if (isset($msg)) { ?>
    window.onload = function() {
      alert("<?php echo $msg ?>");
    }
<?php } ?>
  </script>
 </head>
```

Вызов повторяющегося кода из сценария View

Этот код больше не нужен. Поэтому его можно удалить, а затем вы должны добавить инструкцию require_once, чтобы вызвать ваш новый сценарий с функциями отображения:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

// и т. д...

?>
```

ВНИМАНИЕ

Показать в книге удаленный код практически невозможно. «Посмотрите, это удаленный код. Что? Вы его не видите? Это потому, что он удален!»

Убедитесь в том, что вы удалили из сценария show_users.php тот PHP-код, который вставлялся в head-раздел HTML-вывода для отображения окна предупреждения с вашим сообщением.

Теперь вы можете добавить вызов функции display_messages в свой HTML:

```
<body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пользователи</div>
  <?php display_messages($msg); ?>
```

А теперь нужно решить небольшую проблему: функции display_messages передаются два аргумента: сообщение об успешном завершении и сообщение об ошибке. Следовательно, нужны какие-то средства передачи ей пустого сообщения, а затем функции display_messages нужно будет обработать пустое сообщение, когда оно будет получено.

Вопрос с сообщениями об ошибках решен, они должны стать стандартной частью всех ваших HTML-представлений. Нужно, чтобы при каждом отображении HTML обрабатывались сообщения. И что в таком случае получается?

Вы снова возвращаетесь к повторению кода! Каждый отдельный сценарий, связанный с представлением, начинается с одного и того же базового кода HTML... Хотя временами в него нужно вставлять немного кода JavaScript, как в сценарии show_users.php:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
  <script type="text/javascript">
    function delete_user(user_id) {
      if (confirm("Вы уверены, что хотите удалить этого пользователя?" +
                  "\nВернуть его уже не удастся!")) {
        window.location = "delete_user.php?user_id=" + user_id;
      }
    }
  </script>
 </head>
```

А затем следует тег body, такой же заголовок (еще один пример повторяющегося кода) и заголовок страницы. Теперь у вас есть сообщения для отображения. Появляется еще один шанс взять код, который был набран в ваших сценариях, извлечь этот повторяющийся код и поместить его в дополнительные сервисные функции. Тогда от сценария view.php станет намного больше пользы.

Лучше использовать гибкие функции

Итак, теперь есть нечто вроде списка взаимосвязанностей, с которым нужно работать. Их основная часть связана с обновлением сценария view.php.

- Функция display_messages должна обрабатывать пустые или несуществующие сообщения, касающиеся успешного завершения, а также выводить сообщение

об ошибке. Если какое-либо из сообщений не определено, div-контейнер, относящийся к этому сообщению, не должен выводиться.

- Вам нужна новая функция (назовем ее display_header), которая занимается выводом head-раздела HTML каждой страницы. Этой функции должен передаваться код JavaScript, который должен быть добавлен в head-раздел документа, но она также должна справляться и с теми случаями, когда дополнительный код JavaScript не нужен.
- Необходима еще одна новая функция (дадим ей имя display_title), выводящая заголовок страницы, подзаголовок страницы, который передается ей каждым сценарием, и какое-нибудь сообщение, которое также должно быть передано вызывающим функцию сценарием.

Ни одна из этих функций не отличается особой сложностью, поэтому вернемся к работе.

Использование аргументов по умолчанию в функции display_messages

Возвращаясь к view.php, нужно отметить, что функция display_messages должна быть способна принимать в качестве сообщения отсутствующее значение. В PHP его роль играет специальное ключевое слово NULL, которое просто означает «не значение».

ПРИМЕЧАНИЕ

Ключевое слово NULL можно увидеть почти в любом языке, хотя обычно с небольшими вариациями. В Ruby это слово выглядит как nil. В Java это null. А в PHP — NULL, как и в C++. Эти ключевые слова всегда означают одно и то же: отсутствие значения.

Теперь, поскольку NULL обозначает отсутствующее значение, вы не можете сравнивать его со значением. Поэтому следующий код в PHP не имеет никакого смысла:

```
if ($value == NULL) // какие-нибудь действия
```

Вам нужно воспользоваться еще одной вспомогательной функцией по имени is_null. Вы передаете функции is_null какое-нибудь значение, а PHP дает вам знать, что из этого вышло.

Теперь вы можете обновить функцию display_messages. Если переданное ей сообщение отсутствует (NULL), значит, вызывать отдельную функцию display_message для этого типа сообщения не нужно:

```
function display_messages($success_msg, $error_msg) {
  echo "<div id='messages'>\n";
  if (!is_null($success_msg)) {
    display_message($success_msg, SUCCESS_MESSAGE);
  }
  if (!is_null($error_msg)) {
    display_message($error_msg, ERROR_MESSAGE);
  }
  echo "</div>\n\n";
}
```

Этот сценарий практически доведен до совершенства. Не хватает только одного: что если у такого сценария, как show_users.php, не будет значения, передаваемого переменной $error_msg? Или переменной $success_msg? В таких случаях нужно чтобы у функции display_messages были значения по умолчанию. Это просто значение, присваиваемое переменным на случай, если функции ничего не передано.

Присвоить аргументам функции значения по умолчанию можно следующим образом:

```
function do_something(this_value = "default value") {
  // какие-то действия с this_value
}
```

Итак, для функции display_messages значением по умолчанию должен быть NULL или отсутствие значения:

```
function display_messages($success_msg = NULL, $error_msg = NULL) {
  echo "<div id='messages'>\n";
  if (!is_null($success_msg)) {
    display_message($success_msg, SUCCESS_MESSAGE);
  }
  if (!is_null($error_msg)) {
    display_message($error_msg, ERROR_MESSAGE);
  }
  echo "</div>\n\n";
}
```

Теперь display_messages наконец-то готова выйти на сцену и использоваться другими функциями, которые необходимо добавить к сценарию view.php.

Вывод стандартного заголовка с помощью структуры heredoc

А что же дальше? Нужно поработать со стандартным HTML-выводом для страницы вашего приложения. Обычно это открывающий тег html, тег title, тег head и характерный для страницы JavaScript, который нужно добавить. Имея в своем распоряжении сценарий view.php, сведения о функциях, аргументы по умолчанию и все остальное, что уже сделано, решение этой задачи не составит особого труда.

Можно создать новую функцию. А поскольку вполне возможно, что одним сценариям нужно передавать JavaScript для добавления в head-раздел, а другим нет, здесь опять пригодится в качестве аргумента функции значение по умолчанию:

```
function display_head($page_title = "", $embedded_javascript = NULL) {
```

Этот код устанавливает также значение по умолчанию и для $page_title. В этом нет абсолютной необходимости, но опять же это часть дополнительной подстраховки. Теперь если кто-нибудь, вызывая эту функцию, забудет отправить заголовок, HTML-вывод все равно будет создан.

Телом этой функции будут инструкции echo и условные инструкции для потенциального кода JavaScript:

```
function display_head($page_title = "", $embedded_javascript = NULL) {
  echo "<html>";
```

```
  echo " <head>";
  echo "  <title>{$page_title}</title>";
  echo '  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />';
  if (!is_null($embedded_javascript)) {
    echo "<script type='text/javascript'>" .
         $embedded_javascript .
         "</script>";
  }
  echo " </head>";
}
```

Следует заметить, что в строке с тегом `link` используются одинарные кавычки вокруг HTML, чтобы двойные кавычки можно было применять для атрибутов `href`, `rel` и `type`. К сожалению, приходится либо задействовать несколько видов кавычек, как здесь, либо нейтрализовать кавычки с помощью комбинаций `\"` и `\'`. Ни одно из решений не имеет преимуществ перед другим, поэтому выбор остается за вами.

Разумеется, программисты не привыкли к подобным ограничениям и тут же должны подумать: «Постойте, я же программист. Зачем мне замыкаться на двух не самых лучших решениях?» И по правде говоря, на них действительно не нужно замыкаться. Вам нужен способ работы с многострочными фрагментами вывода, и PHP в данном случае вас не разочарует. Многострочные фрагменты являются довольно распространенной задачей в PHP, решаемой двумя способами.

Чаще всего используется структура, называемая heredoc (указатель на присутствие документа). Структура heredoc предоставляет вам способ, позволяющий пометить начало и конец текстового фрагмента. Затем все, что находится между этими метками, рассматривается в качестве текста, который не нужно окружать чем-нибудь вроде кавычек.

Начало фрагмента heredoc обозначается тремя символами «меньше чем», а затем указывается последовательность, которая используется для обозначения конца фрагмента:

```
$some_text = <<<EOD
```

Итак, здесь говорится: «Я начинаю некий текст, который закончится там, где вам встретится аббревиатура `EOD`».

ПРИМЕЧАНИЕ

Можно использовать любую завершающую последовательность. Но чаще всего выбираются аббревиатуры EOD и EOT, поэтому лучше остановиться на них, если у вас не будет более веских причин для применения другой последовательности.

А теперь вы можете поместить в эту структуру столько текста, сколько вам нужно. Можно использовать несколько строк, одинарные, двойные кавычки и даже синтаксис `{$var_name}`. Все это будет в пределах правил:

```
<html>
 <head>
  <title>{$page_title}</title>
   <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>
```

И наконец, текст нужно закончить закрывающей последовательностью:

```
EOD;
```

Если все это объединить, получится следующий код:

```
$some_text = <<<EOD
<html>
 <head>
  <title>{$page_title}</title>
   <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>
EOD;
```

ВНИМАНИЕ

Завершающая последовательность не должна иметь никакого отступа. Она должна быть первым и единственным элементом в строке, и перед ней не должно быть никаких пробелов.

Точно так же опасны и пробелы после завершающей последовательности. Проиллюстрировать это невозможно, но даже единственный пробел после закрывающей точки с запятой может привести к плачевным последствиям.

Обнаружить подобные просчеты лучше всего путем отслеживания такой грозной ошибки unexpected T_SL. Это обычный, но достаточно таинственный способ, которым PHP оповещает о наличии пробельных символов там, где их быть не должно: чаще всего перед или после завершающей последовательности.

Соберите все это вместе, и тогда в функции display_head станет проще разобраться:

```
function display_head($page_title = "", $embedded_javascript = NULL) {
  echo <<<EOD
<html>
 <head>
  <title>{$page_title}</title>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
EOD;
  if (!is_null($embedded_javascript)) {
    echo "<script type='text/javascript'>" .
         $embedded_javascript .
         "</script>";
  }
  echo " </head>";
}
```

Вы, наверное, заметили, что в этой версии display_head не понадобилось присваивать переменной строковое значение, созданное с помощью heredoc. Многострочную структуру можно вывести непосредственным образом, исключив тем самым лишний этап. В результате получается смесь из инструкции echo, структуры heredoc, условной логики и потенциально возможного кода JavaScript. Но код становится удобнее для чтения, и это неплохой результат.

Обновление сценария (сценариев) для использования функции display_head

Настало время объединить разрозненные фрагменты выводимой информации в единое целое. Теперь можно вернуться к show_users.php (и при необходимости к show_user.php) и удалить множество фрагментов HTML. Замените HTML, предназначенный для заголовка вашего документа, вызовом функции display_head. Занимаясь сценариями, можете в процессе работы еще раз воспользоваться структурой heredoc, особенно это касается сценария show_users.php, который отправляет для встраивания в страницу фрагмент кода JavaScript:

```
<?php
// код для получения всех данных о пользователе
?>

<?php
  $delete_user_script = <<<EOD
function delete_user(user_id) {
  if (confirm("Вы уверены, что хотите удалить этого пользователя?" +
                "Вернуть его уже не удастся!")) {
    window.location = "delete_user.php?user_id=" + user_id;
  }
}
EOD;
  display_head("Пользователи", $delete_user_script);
?>

<!-- Вся остальная HTML-разметка -->
</html>
```

ПРИМЕЧАНИЕ

Вы можете поместить весь PHP, который получает данные о пользователях, в блок <?php/?> (как и код, вызывающий display_head). Все зависит от вас. Некоторые программисты предпочитают содержать код, собирающий данные, отдельно от кода, который выводит представление, а некоторые предпочитают не дублировать <?php. Выбор за вами.

Здесь структура heredoc используется с той целью, чтобы при создании строки JavaScript, передаваемой функции display_head, не приходилось заниматься многочисленной нейтрализацией одинарных и двойных кавычек. Структура heredoc практически не менее полезна, чем функция sprintf. И вы без ограничений можете пользоваться обоими средствами для вывода HTML или других длинных отрезков текста.

Остается еще проблема вывода сообщений. Но прежде чем приступить к ее решению, проверьте работу измененного кода сценария show_users.php. Вы должны увидеть что-либо похожее на изображение, показанное на рис. 10.16.

Таким образом, это еще один случай, когда проводится довольно большой объем работы в надежде на то, что результаты будут иметь точно такой же вид, как



Рис. 10.16. Результат выполнения измененного сценария show_users.php

и прежде. Но теперь используются функции в view.php, а не сам выводимый HTML. Получаемый в результате заголовок должен выглядеть точно так же, как и заголовок любой другой страницы... поскольку для всех этих страниц теперь используется функция display_head.

Стандартизация и объединение вывода сообщений в сценарии View

Теперь осталось только разобраться с сообщениями. У вас есть функция display_messages, но она не встроена в тот HTML, который обычно окружает такие сообщения. Точно так же, как функция display_head выводит HTML с потенциально встроенным кодом JavaScript, в первой части вашей страницы должны выводиться некий стандартный HTML, заголовок страницы (еще раз), потенциальное сообщение об успешном завершении операции и сообщение об ошибке. Поэтому окончательно код, предназначенный для вывода информации, должен иметь следующий вид:

```
<html>
 <head>
  <title>Пользователи</title>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
  <script type='text/javascript'>function delete_user(user_id) {
  if (confirm("Вы уверены, что хотите удалить этого пользователя?" +
                "Вернуть его уже не удастся!")) {
    window.location = "delete_user.php?user_id=" + user_id;
  }
}</script>
```

```
</head>
 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">Пользователи</div>
  <div id='messages'>
 <div class='success'>
  <p>Указанный пользователь был удален.</p>
 </div>
</div>

  <div id="content">
    <!-- HTML-содержимое -->
  </div>
 </body>
</html>
```

Теперь все будет еще проще. Создайте в сценарии view.php функцию display_title:

```
function display_title($title, $success_msg = NULL, $error_msg = NULL) {
echo <<<EOD
 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">$title</div>
EOD;
  display_messages($success_msg, $error_msg); ?>
}
```

Ну что, все просто? Теперь эту функцию можно вызвать, скажем, из show_users.php следующим образом:

```
display_title("Пользователи", $msg);
```

Но вы уже знаете, как передаются сообщения: доступ к ним осуществляется через параметры запроса путем использования переменной $_REQUEST. Поэтому зачем волноваться о том, установлены они в вашем представлении или нет? Для отображения заголовка просто передайте их представлению, даже если значение будет равно NULL:

```
display_title("Пользователи",
              $_REQUEST['success_message'], $_REQUEST['error_message']);
```

ПРИМЕЧАНИЕ

Вы можете также удалить в show_users.php код, который получает для функции success_message параметры запроса непосредственно из переменной $_REQUEST, поскольку этот процесс теперь осуществляется нашим новым вызовом функции display_title.

Все выглядит весьма неплохо. Обе функции display_head и display_title готовы к работе, и вызовы функции display_head уже присутствуют в коде.

Но перед добавлением вызова функции display_title во всех ваших сценариях обдумайте все предстоящие действия.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Почему бы не передать $_REQUEST в display_title?

Вы можете подумать о том, что можно было бы целиком передать переменную `$_REQUEST` функции `display_title`. Затем `display_title` могла бы извлечь значения `$_REQUEST['success_message']` и `$_REQUEST['error_message']`. Это неплохая идея. Ее конкретный смысл заключается в том, что ваши сценарии отображения не будут заботиться о том, где какой параметр запроса, или даже о том, какие конкретно параметры запроса поступили.

Потенциальный недостаток состоит в том, что код представления (функции в сценарии `view.php`, которые заняты в основном выводом кода HTML) привязывается к способу получения данных для этого представления. Теперь ваше представление взаимодействует с самим пользовательским запросом вместо того, чтобы давать возможность заниматься этим контроллеру, который передает информацию по мере необходимости.

Вы еще раз можете убедиться в том, что внедрить MVC-архитектуру в PHP просто невозможно. Вам придется постоянно принимать компромиссные решения между четким разделением представления и контроллера, а также простотой программирования. В данном случае можно все оставить как есть и позволить сценарию `view.php` просто выводить информацию или же дать ему небольшую дополнительную нагрузку и передать этому сценарию переменную `$_REQUEST`.

Создание функции для вызова двух функций

Теперь вспомним, что замысел состоял в создании еще одной функции — `display_title`, предназначенной для вывода начального фрагмента `body`-раздела каждой HTML-страницы. Теперь, когда у вас есть эта функция, нужно подумать вот о чем:

- за кодом HTML из функции `display_title` будет всегда непосредственно следовать HTML-вывод из функции `display_head`;
- заголовок, используемый в функции `display_head`, должен, как правило, соответствовать заголовку, который применяется в функции `display_title`.

Если этот HTML всегда следует за HTML из функции `display_head` и заголовок в обеих функциях один и тот же, зачем тогда эти два вызова? В ваших сценариях у вас всегда будет примерно следующий код:

```
<?php

// Какой-нибудь невероятный код

?>

<?php display_head($title, $javascript);  ?>
<?php display_title($title,
```

```
                  $_REQUEST['success_message'], $_REQUEST['error_message']);
?>

 <!-- Дальнейший код HTML -->
</html>
```

Но нужно ли это? Зачем здесь два вызова? Нельзя ли навести здесь порядок?

```
<?php

// Какой-нибудь невероятный код

?>

<?php page_start($title, $javascript,
                 $_REQUEST['success_message'], $_REQUEST['error_message']) ?>

 <!-- Дальнейший код HTML -->
</html>
```

Теперь не только упростился вызов, но и исчезла необходимость в двойной передаче переменной `$title`. Это делается один раз и применяется во всем открытом HTML-коде.

Теперь вам не нужно крутиться между `display_title`, `display_head` или `display_messages`. Вместо этого следует просто создать функцию для вашего сценария, вызываемую для работы с этими более мелкими по назначению функциями:

```
function page_start($title, $javascript = NULL,
                    $success_message = NULL, $error_message = NULL) {

  display_head($title, $javascript);
  display_title($title, $success_message, $error_message);
}
```

ПРИМЕЧАНИЕ

Поместите эту функцию в сценарий view.php вместе со всеми вашими другими функциями, занимающимися отображением данных.

Отлично! Теперь обо всем этом позаботится один вызов из сценария представления.

Теперь нужно просто распространить эту информацию на весь код

Что еще осталось сделать? Удалить вызовы функции `display_head`, исключить другие вызовы `display_title` и в заключение сделать один вызов для управления всеми этими функциями.

ПРИМЕЧАНИЕ

Это была шутка в стиле «Властелина колец». Но за 380 страниц изучения программирования на PHP вы заслужили такую злую шутку.

На самом деле нужно посмотреть на новую улучшенную версию сценария show_users.php. Он стал короче и понятнее. Даже при том небольшом беспорядке в отступах, привносимом структурой heredoc, этот сценарий выглядит совсем неплохо:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

// Создание инструкции SELECT
$select_users =
  "SELECT user_id, first_name, last_name, email " .
  "  FROM users";

// Запуск запроса
$result = mysql_query($select_users);

// Отображение представления пользователям
  $delete_user_script = <<<EOD
function delete_user(user_id) {
  if (confirm("Вы уверены, что хотите удалить этого пользователя?" +
                "Вернуть его уже не удастся!")) {
    window.location = "delete_user.php?user_id=" + user_id;
  }
}
EOD;
  page_start("Current Users", $delete_user_script,
             $_REQUEST['success_message'], $_REQUEST['error_message']);
?>
  <div id="content">
   <ul>
     <?php
       while ($user = mysql_fetch_array($result)) {
         $user_row = sprintf(
           "<li><a href='show_user.php?user_id=%d'>%s %s</a> " .
           "(<a href='mailto:%s'>%s</a>) " .
           "<a href='javascript:delete_user(%d);'><img " .
              "class='delete_user' src='../images/delete.png' " .
              "width='15' /></a></li>",
           $user['user_id'], $user['first_name'], $user['last_name'],
           $user['email'], $user['email'], $user['user_id']);
         echo $user_row;
       }
     ?>
   </ul>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

Теперь нужно развить успех. Убедитесь, что работает вывод сообщений об ошибках и об успешном завершении операции. Внесите изменения в другие свои сценарии, чтобы в них также использовалась функция page_start. Добавьте еще какие-нибудь функции к view.php. Возможно, вам захочется создать функцию page_end, которая будет выводить завершающий страницу div-контейнер с идентификатором, который имеет значение footer, и какой-нибудь контактной информацией. Можно также добавить функцию, создающую боковую панель.

Ведь с таким модульным подходом можно делать все, что угодно... за исключением управления правами удаления пользователей. Эта задача будет решаться в следующей главе.

ЗАМЕТКИ О ПРОЕКТИРОВАНИИ

В каком-то смысле две функции лучше одной

Вам все чаще и чаще приходится сталкиваться с идеей перемещения все меньших и меньших фрагментов кода в отдельные функции. Таким образом у вас появился небольшой фрагмент HTML в функции сценария view.php, а также сценарий database_connection.php, осуществляющий подключение к базе данных. И даже притом, что этот код не был выделен в пользовательскую функцию, он все же вызывается как функция с помощью инструкции require_once. То же самое имело место еще несколько раз: брались небольшие части каких-то действий или функциональных средств и помещались в небольшие, доступные для вызова функции.

Поэтому можно было прийти к выводу, что цель состоит в создании большого количества вызовов отдельных функций. Частично так и есть. Но на самом деле это продиктовано желанием получить множество строительных блоков, которые можно будет собирать в более крупные полезные части. Но когда дело доходит до использования таких функций, неужели вам захочется сделать 20 или 30 отдельных вызовов?

Наверное, нет.

Скорее всего, вместо этого вы пожелаете свести количество вызовов в своем сценарии к минимально необходимому значению. По крайней мере ограничиться теми вызовами, которые необходимы для взаимодействия с пользователем. Поэтому предпочтительнее будет осуществлять вызов примерно так:

```
display_page($title, $javascript, $content);
```

нежели так:

```
display_head($title, $javascript);
display_messages($msg);
display_content($content);
display_footer();
```

Конечно, при этом не нужно все переворачивать с ног на голову и сводить весь код из десяти функций в одну. Но может так сложиться, что вам захочется иметь одну функцию, которая затем вызовет для вас все эти функции. При этом будут

по-прежнему использоваться строительные блоки, но сократится количество действий, которые нужно будет выполнять для нормальной работы в ваших сценариях верхнего уровня.

Подумайте о том, не проще ли будет не забыть вызвать функцию `display_page`, а затем найти аргументы для передачи? Или проще не забыть вызвать `display_head`, а затем `display_messages`, а потом `display_content`, после чего... какая там следующая функция? Разумеется, проще вызвать одну функцию.

Именно поэтому появляется желание двигаться в направлении создания группы из разных небольших функций или создания функций более высокого уровня, объединяющих эти небольшие функции каким-то полезным образом. В ваших сценариях должны быть такие вот простые вызовы, способные на необходимые вам действия, даже если под ними подразумевается вызов большого количества меньших по объему функций.

В результате должен получиться более простой и легче читаемый код. Но, кроме того, вы у вас будет в распоряжении великолепный набор функций, которые можно будет сочетать друг с другом наиболее полезными и разнообразными способами.

ЧАСТЬ 4

Безопасность и реальное окружение

Глава 11. Аутентификация и авторизация

Глава 12. Cookie-файлы, вопросы регистрации и избавление от примитивных окон

Глава 13. Авторизация и сессии

11 Аутентификация и авторизация

В дизайне и в создании вашего приложения на данный момент происходит что-то странное. В вашем распоряжении имеется четыре, пять, а может быть и больше свойств. Есть также несколько таблиц. Создано уже множество внутренних механизмов приложения, но сохранена простота всех составляющих, и вам понятно, что и откуда берется.

Добавлены также некоторые новые свойства, например возможность удаления пользователей. Это уже похоже на еще одно свойство, напротив названия которого в списке можно поставить галочку. Но постойте... удаление пользователей? Неужели такое мощное свойство можно давать в руки всем вашим пользователям? Конечно же, нет. Это свойство для администратора.

ПРИМЕЧАНИЕ

Возможно, вы помните, что сначала сценарий delete_user.php мы хотели назвать admin.php.

Но кто такой администратор? Очевидно, что это человек или группа людей, управляющих учетными записями; наверное, это кто-то, у кого на мониторе приклеено еще несколько стикеров с паролями. Но в вашем приложении пока нет администратора. В данный момент любой может добраться до сценария `delete_user.php` и удалить бедного пользователя по имени Уильям Шетнер или Джеймс Родей либо какую-нибудь другую знаменитость, зарегистрировавшуюся через `create_user.html`, а также его друзей.

Но дело обстоит куда хуже! Поскольку небольшие красные крестики появляются при запуске сценария `show_users.php`, кто-нибудь, просто рассматривающий пользователей, вдруг может заметить маленький крестик, с помощью которого можно удалить данные навсегда. И не имея на своем пути никаких других препятствий, кроме окна подтверждения, может получить доступ к этой функциональной возможности.

В жизни вашего приложения наступает знаменательный момент. Вы добавили одну функциональную возможность, но пришли к пониманию, что при этом необходимо реализовать какие-то другие свойства, которые вскоре понадобятся. Чтобы пользователи получили работоспособное приложение, нужно решить несколько задач, перечисленных в следующем списке.

- Просмотр всех пользователей (эта задача уже решена).
- Удаление пользователей (уже решена, но с излишне свободным доступом к данной функции).
- Разработка способа идентификации пользователей в системе (отчасти, он уже есть в сценарии `create_user.html`, но пока в нем нет входа в систему).
- Разработка способа, позволяющего определить наличие у данного пользователя прав администратора.
- Разработка способа входа пользователей в систему и проверки, кем они являются (например, с помощью пароля).
- Разработка способа показа только определенных функциональных возможностей, например возможности удаления пользователя, если пользователь является администратором.

В общем, вашей системе не хватает аутентификации. Пользователь должны регистрироваться в системе, после чего система будет знать, относится ли пользователь к той или иной категории, например к администраторам. А затем на основании принадлежности к той или иной категории пользователь будет видеть (или не видеть) конкретные элементы. Такая выборочная демонстрация ресурсов или выборочное недопущение вообще ни к каким ресурсам является аутентификацией (или авторизацией), направленной против злоумышленников.

Следовательно, вы проходите аутентификацию и даете системе понять, кто вы такой. И на основании вашей принадлежности к той или иной категории вы авторизуетесь на просмотр конкретных элементов. Аутентификацию и авторизацию часто путают между собой.

ПРИМЕЧАНИЕ

Некоторые считают позором путать аутентификацию с авторизацией. Но у таких людей, наверное, есть отдельные выдвижные ящички для носков каждого цвета. Понимать, в чем разница между этими понятиями, конечно, хорошо, но зацикливаться на мелочах не стоит.

Нет ни малейших сомнений, что нужно добавить аутентификацию и авторизацию в ваше растущее в объеме приложение. Вы регистрируетесь почти на каждом регулярно посещаемом сайте. Даже у таких сайтов, как YouTube и Google, есть регистрация, и, конечно же, она есть в Twitter и Facebook. Все эти сайты используют аутентификацию, чтобы знать, кто есть кто. Пора бы и вашему приложению присоединиться к этой компании.

Начнем со стандартной аутентификации

Аутентификация, как и любой другой процесс, может проводиться простым или очень сложным способом. Как обычно, лучше начинать с основ и добавлять сложность по мере необходимости. Для простого приложения не требуются системы

чтения отпечатков пальцев и лазерного сканирования лиц ваших пользователей. (Конечно, было бы забавно применять их, но в этом нет необходимости. Скорее всего, Джеймс Бонд не станет заполнять вашу форму create_user.html.)

Стандартная аутентификация с использованием HTTP-заголовков

Стандартная аутентификация, известная также как HTTP-аутентификация, означает получение в веб-приложении имени пользователя и пароля через HTTP-заголовки. С заголовками вам уже приходилось работать. Помните этот фрагмент кода:

```
function handle_error($user_error_message, $system_error_message) {
  header("Location: " . get_web_path(SITE_ROOT) .
         "scripts/show_error.php" .
         "?error_message={$user_error_message}" .
         "&system_error_message={$system_error_message}");
}
```

ПРИМЕЧАНИЕ

Функция handle_error находится в файле scripts/app_config.php.

В этом коде нет ничего, кроме HTTP-заголовка, то есть заголовка Location для отправки браузеру перенаправления. При показе изображения вам также приходилось использовать заголовки Content-type и Content-length:

```
header('Content-type: ' . $image['mime_type']);
header('Content-length: ' . $image['file_size']);
```

ПРИМЕЧАНИЕ

Этот код был частью сценария show_image.php из главы 9, который использовался для показа изображения, сохраненного в базе данных.

При стандартной аутентификации можно отправить два других HTTP-заголовка. У первого нет ключевого значения вроде Content-type или Location. Необходимо просто отправить сам заголовок:

```
HTTP/1.1 401 Unauthorized
```

Когда браузер получает этот заголовок, он знает, что страница для своего отображения затребовала аутентификацию. 401 — это специальный код состояния, один из многих других, который сообщает браузеру информацию о запросе. Код 200, например, используется, чтобы сообщить, что все в порядке, а 404 является кодом ошибки HTTP.

ПРИМЕЧАНИЕ

Подробнее о кодах состояния HTTP можно прочитать на сайте www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

Сообщение браузеру о том, что доступ к странице ограничен, проблему не решает. А как получить эту страницу без ограничений? Нужно отправить второй заголовок:

```
WWW-Authenticate: Basic realm="The Social Site"
```

Он оповещает браузер о необходимости аутентификации. А именно о том, что браузер должен вывести окно и запросить полномочия.

Здесь используется заголовок `WWW-Authenticate`, и браузеру сообщается, какой тип аутентификации нужен — `basic`. Затем дается область (`realm`), к которой должна быть применена аутентификация. В данном случае это `"The Social Site"`. Из этого следует, что пока различные страницы используют эту же самую область, аутентификация по отношению к одной из таких страниц будет применяться к другим страницам, находящимся в этой же области.

Стандартная аутентификация проводится... стандартно

Чтобы увидеть, как работает стандартная аутентификация, попробуйте добавить ее в сценарий `show_users.php`. Введите следующие две строки с заголовками где-нибудь в верхней части сценария:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

header('HTTP/1.1 401 Unauthorized');
header('WWW-Authenticate: Basic realm="The Social Site"');

// Создание инструкции SELECT
$select_users =
  "SELECT user_id, first_name, last_name, email " .
  "  FROM users";

// Весь остальной код PHP

?>
```

ПРИМЕЧАНИЕ

Как и обычно, необходимо подумать о создании резервной копии этого сценария или же нужно скопировать все ваши сценарии в новый каталог ch11. Тогда у вас будут все старые, работоспособные сценарии, к которым можно будет вернуться, если что-нибудь пойдет не так.

Теперь перейдите к сценарию `show_users.php`. Вы должны увидеть красивое окно, которое будет запрашивать у вас регистрацию (рис. 11.1). Ну, не такое уж оно и красивое, но со своей ролью справляется. В конце концов, это ведь не что иное, как простая стандартная аутентификация.

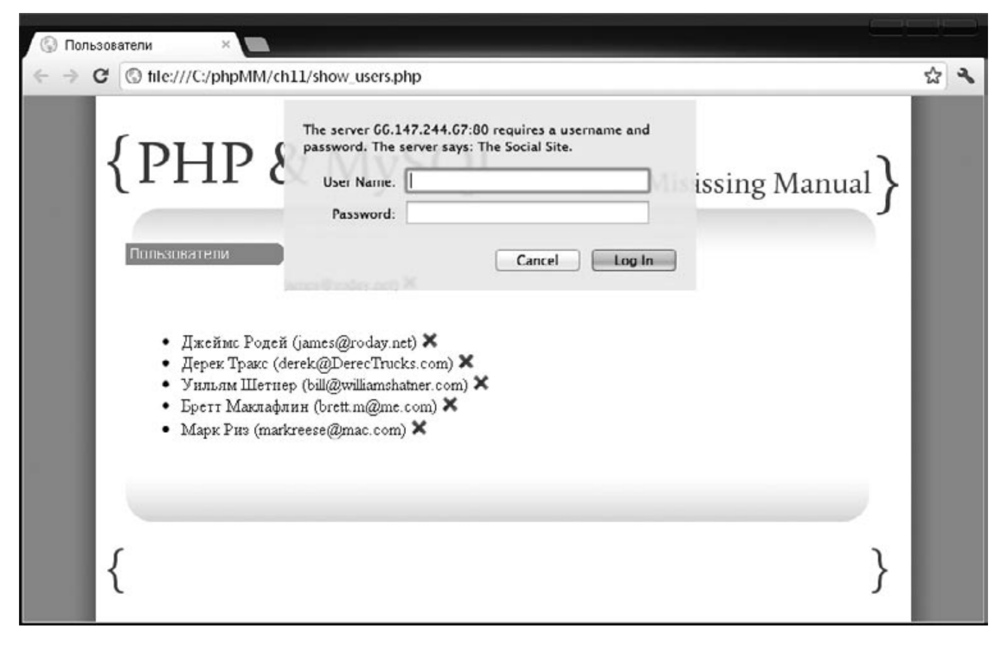


Рис. 11.1. Стандартная аутентификация

ВНИМАНИЕ

Если на вашем веб-сервере для ограничения сетевого доступа к конкретным каталогам используется файл .htpasswd (особенно популярный на веб-серверах Apache), то могут возникнуть проблемы. Файл .htpasswd иногда не хочет нормально работать со стандартной аутентификацией PHP. Лучше всего будет позвонить провайдеру и просто попросить его не использовать никаких файлов .htpasswd в отношении каталогов, с которыми вы работаете.

Самая худшая из всех аутентификаций

Однако пока в вашей системе безопасности просматривается огромная дыра. Перейдите на страницу сценария show_users.php, если вы еще не на ней, и оставьте пустыми оба поля — и для имени пользователя, и для пароля. Затем просто щелкните на кнопке Cancel (Отмена). Что получится в результате? То, что показано на рис. 11.2.

Но это еще не все, введите в поля любое имя пользователя и пароль и щелкните на кнопке регистрации. И вы опять попадете на страницу, показанную на рис. 11.2. Любые попытки получить что-либо другое, кроме страницы, выводимой сценарием show_users.php, будут безрезультатны.

О какой безопасности здесь вообще может идти речь? Очевидно, что-то не в порядке. При отмене работы с окном регистрации вы не должны попадать на страницу, которая считается защищенной. Вам нужно получить введенные имя пользователя и пароль, проверить их соответствие приемлемым значениям, а затем показать страницу. В любом другом случае пользователь не должен видеть страницу, создаваемую сценарием show_users.php.

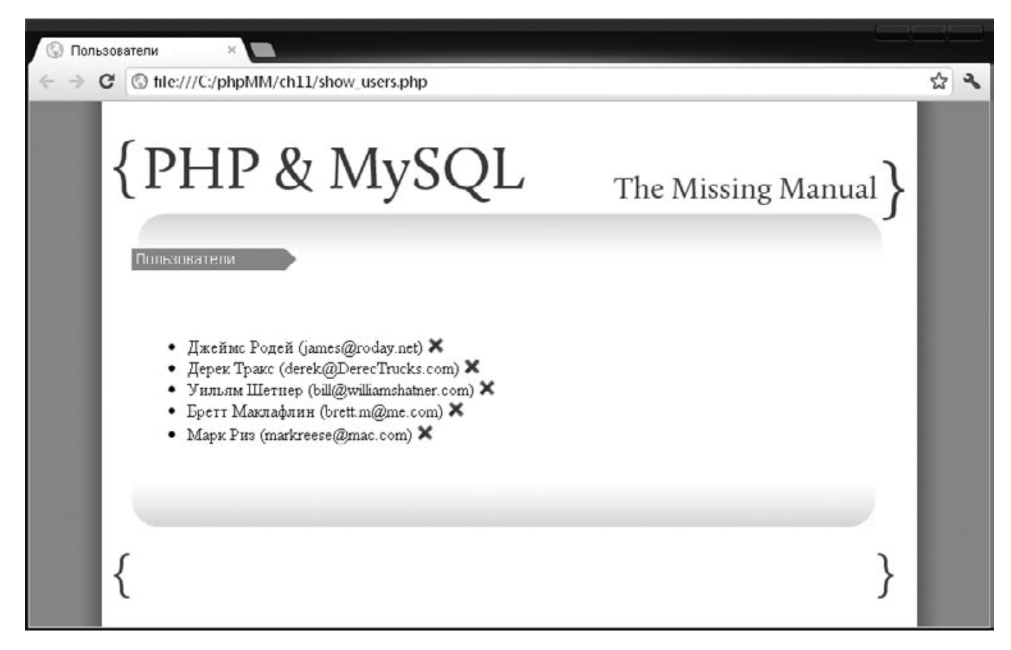


Рис. 11.2. После отмены в окне регистрации выводится страница сценария show_users.php

Получение данных о полномочиях вашего пользователя

К сожалению, сколько бы вы ни старались, пока в сценарий не будут внесены изменения, вам не удастся сопоставить имя пользователя и пароль с какими-нибудь значениями. Ведь ваш код даже не извлекает эти значения, чтобы позволить сравнить их с какими-нибудь другими значениями, которые подойдут для просмотра страницы, создаваемой сценарием `show_users.php`. Очевидно, придется немного потрудиться.

Но поскольку HTTP-аутентификация считается стандартным способом, установить с помощью кода PHP взаимодействие с пользователем, который вводит свое имя пользователя и пароль в стандартное появляющееся окно аутентификации, не составляет особого труда. Интерпретатор PHP предоставляет вам доступ как к введенному имени пользователя, так и к паролю. Для этого используются два специальных значения в сверхглобальной переменной `$_SERVER`, которой вы уже пользовались:

- `$_SERVER['PHP_AUTH_USER']` — предоставляет введенное имя пользователя;
- `$_SERVER['PHP_AUTH_PW']` — предоставляет введенный пользователем пароль.

ПРИМЕЧАНИЕ

Переменная $_SERVER используется для определения константы SITE_ROOT в сценарии app_config.php, а также в сервисной функции get_web_path.

Теперь можно подумать, что действия должны развиваться следующим образом.

1. В начале сценария нужно отправить HTTP-заголовки, инициирующие аутентификацию.
2. После завершения работы кода аутентификации нужно проверить наличие значений в переменных `$_SERVER['PHP_AUTH_USER']` и `$_SERVER['PHP_AUTH_PW']` и сравнить эти значения с некими константами или информацией из базы данных.
3. Нужно решить, следует позволить пользователю увидеть обычный вывод сценария или нет.

Вполне разумно, но, как выясняется, неверно... в корне неверно.

Вот что получается.

1. Вызывается сценарий.
2. Отправляются заголовки (имеются в виду те, которые говорят о том, что пользователь не авторизован и ему нужно дать возможность зарегистрироваться).
3. Как только пользователь вводит имя пользователя и пароль, браузер снова вызывает ваш сценарий и он опять выполняется с самого начала.

Поэтому перед отправкой заголовков аутентификации нужно посмотреть, есть ли у пользователя какие-нибудь приемлемые полномочия. А затем, если таковые имеются, сравнить их с разрешенными значениями. И наконец, если полномочия не соответствуют или их нет, отправить заголовки аутентификации.

И здесь опять вам на помощь приходит функция `isset`. Начните примерно со следующего кода:

```
if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW'])) {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
}
```

Но этот код всего лишь выводит окно регистрации, если код пользователя и пароль не были заданы. Это по-прежнему позволяет получить доступ к вашей странице несколькими различными способами. Поэтому вам нужно не только вывести на экран окно регистрации, но и позаботиться о том, чтобы любые предварительно введенные имена пользователей и пароли соответствовали набору разрешенных значений.

Отмена не подходит для аутентификации

Но перед тем, как заняться проверкой имени пользователя и пароля, есть более серьезная работа: принятие щелчка на кнопке отмены программируется еще сложнее, чем принятие любого имени пользователя и пароля.

Разобраться с самим щелком на кнопке отмены довольно просто, хотя используемый способ не столь очевиден. Вот как выглядит код на данный момент:

```
if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW'])) {
```

```
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
}
```

Окно регистрации вызывается путем двух вызовов функции header:

```
header('HTTP/1.1 401 Unauthorized');
header('WWW-Authenticate: Basic realm="The Social Site"');
```

Когда пользователь щелкает на кнопке отмены, код PHP продолжает выполняться именно с той строки, которая следует за вторым вызовом функции header:

```
header('HTTP/1.1 401 Unauthorized');
header('WWW-Authenticate: Basic realm="The Social Site"');
// Эта строка выполняется в том случае, если на кнопке отмены
// был выполнен щелчок
```

В самом простом варианте можно просто выйти из сценария:

```
if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW'])) {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
}
```

Итак, если теперь пользователь щелкнет на кнопке отмены, сценарий запустит команду exit, которая во многом похожа на команду die, и произойдет выход из сценария с выдачей сообщения об ошибке (рис. 11.3).

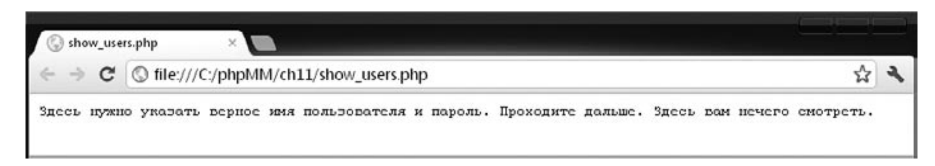


Рис. 11.3. Если пользователь попытается «схитрить» и щелкнуть на кнопке отмены без прохождения регистрации, ваш сценарий справится с ситуацией

Получение пользовательских полномочий (на этот раз всерьез!)

Теперь вы можете вернуться назад, чтобы посмотреть, что ваш пользователь ввел в окне регистрации. Следует помнить, что ход выполнения сценария здесь может быть не таким, как вы могли ожидать. Как только пользователь вводит имя пользователя и пароль, сценарий вызывается заново. Это почти то же самое, как если бы сервер сам предоставил вам цикл while, имеющий примерно следующий вид:

```
while (даны_неверные_имя_пользователя_и_пароль) {
  еще_раз_запросить_имя_пользователя_и_пароль();
}
```

ПРИМЕЧАНИЕ

Этот сценарий не является запускаемым, работоспособным кодом PHP. Это нечто, называемое псевдокодом. Дополнительные сведения о псевдокодах даны в следующей врезке.

Теперь можно посмотреть, установлены ли имя пользователя и пароль. Если нет, нужно отправить заголовки, а если был щелчок на кнопке отмены, — выйти из сценария.

```
if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW'])) {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
}
```

К ВАШЕМУ СВЕДЕНИЮ

Псевдокод является предшественником настоящего кода

Довольно часто возникает недостаток в каком-нибудь промежуточном средстве, предшествующем написанию полноценного работоспособного кода с нормальным синтаксисом и возможностью отладки и запуска, которое могло бы послужить для набросков замысла в ноутбуке. Вам хочется продумать особенности того, как все будет работать, не вдаваясь при этом в какие-то мелочи.

Именно для таких ситуаций и был придуман псевдокод. Как и в настоящем коде, в нем обычно используется синтаксис языка, на котором ведется работа. То есть вы можете использовать `if`, `while`, `else`, а то, что со временем должно будет превратиться в код PHP, обычно помещается в фигурные или угловые скобки. Поэтому следующие строки:

```
while (даны_неверные_имя_пользователя_и_пароль) {
  еще_раз_запросить_имя_пользователя_и_пароль();
}
```

могут послужить хорошим примером псевдокода, который позже станет кодом PHP. В показанном выше примере нет смысла набирать все, что связано с переменной `$_SERVER`, поскольку это будет слишком длинно, а основная идея понятна и без этого. Поэтому, объясняя сотруднику, что именно создается, или просто планируя код, вполне достаточно остановиться на следующем:

```
while (даны_неверные_имя_пользователя_и_пароль) {
```

А представлять при этом что-нибудь следующее:

```
if (($_SERVER['PHP_AUTH_USER'] !=
      VALID_USERNAME) ||
    ($_SERVER['PHP_AUTH_PW'] !=
      VALID_PASSWORD)) {
```

А что вы будете делать после данного определения? Что-нибудь... Что именно, вы пока не в курсе. Вы в общих чертах знаете, что должно случиться, но подробностей еще нет. Остается только определить следующее:

```
еще_раз_запросить_имя_пользователя_и_пароль();
```

Здесь все ясно и понятно, без лишней путаницы. Это псевдокод. Он хорош для развития идеи или обсуждения кода. Он также хорош в ситуациях, подобных этой, когда что-то подсказывает вам, что используемый вами способ решения задач может потребовать изменений. И когда наступит время для этих изменений, то чем меньше работы будет вложено в то решение, которое не будет взято за основу, тем лучше.

В остальной части этого сценария, которую еще нужно написать, вы должны проверить имя пользователя и пароль на приемлемость значений. Если значения совпадут с допустимыми, код будет выполняться дальше и покажет вывод, создаваемый сценарием show_users.php. Если значения не совпадут, то нужно будет снова отправить заголовки, заставляющие браузер еще раз предложить пользователю зарегистрироваться. То есть вам нужен примерно следующий код:

```
if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW'])) {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
} else {
  if (($_SERVER['PHP_AUTH_USER'] != VALID_USERNAME) ||
      ($_SERVER['PHP_AUTH_PW'] != VALID_PASSWORD)) {
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
  }
}
```

ПРИМЕЧАНИЕ

С технической точки зрения блок if предоставляет функции exit неверное сообщение. Эта функция exit имеет дело с тем случаем, когда пользователь щелкнул на кнопке отмены, а не ввел неверное имя пользователя и пароль. Но, как правило, при срабатывании системы безопасности пользователю предоставляется минимальный объем информации. Поэтому лучше применить обобщенное сообщение «на все случаи жизни».

Здесь можно кое-что объединить. Если пользователь даже не попытался зарегистрироваться или неправильно ввел свое имя или пароль, сценарий должен отправить HTTP-заголовки для принудительной аутентификации. Все остальные действия на странице, включая вывод информации, будут предприняты только в том случае, если пользователь указал информацию и она совпала с приемлемыми значениями. Поэтому вам нужен следующий код:

```
if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW']) ||
    ($_SERVER['PHP_AUTH_USER'] != VALID_USERNAME) ||
    ($_SERVER['PHP_AUTH_PW'] != VALID_PASSWORD)) {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
}
```

ПРИМЕЧАНИЕ

Добавьте этот код к своей версии сценария show_users.php.

Теперь перейдите в начало сценария show_users.php, установив курсор до новой инструкции if, и добавьте константы:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

define(VALID_USERNAME, "admin");
define(VALID_PASSWORD, "super_secret");

if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW']) ||
    ($_SERVER['PHP_AUTH_USER'] != VALID_USERNAME) ||
    ($_SERVER['PHP_AUTH_PW'] != VALID_PASSWORD)) {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
}
```

Попробуйте зайти на страницу сценария show_users.php еще раз и введите в качестве имени пользователя и пароля admin и super_secret (рис. 11.4). Вас поприветствует нормальный вид страницы show_users.php (рис. 11.5). В противном случае вы будете снова и снова получать окно аутентификации (см. рис. 11.1).



Рис. 11.4. Браузер отвечает на ваши заголовки окном регистрации и передает значения интерпретатору PHP посредством сверхглобальной переменной $_SERVER

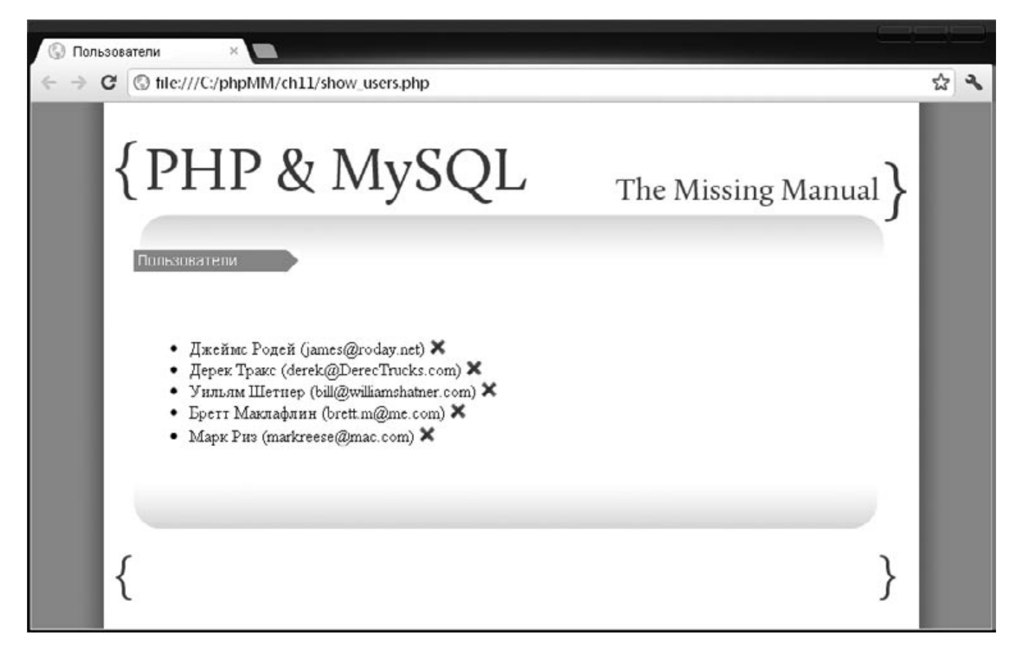


Рис. 11.5. Миновав систему безопасности, вы возвращаетесь к просмотру пользователей; аутентификация отделена от основного содержимого ваших страниц

Извлечение всего одинакового

И опять в сценарии show_users.php оказался код, который ему не должен принадлежать. Почему так получилось? Потому что такая авторизация и аутентификация, которая имеется в сценарии show_users.php, должна быть принадлежностью любого другого сценария, который должен требовать прохождение регистрации, например сценария delete_user.php. Не нужно набирать этот код снова и снова. С ним следует поступить так же, как и с любым другим повторяющимся кодом, который находится в сценариях app_config.php и database_connection.php: его необходимо переместить в отдельные сценарии, а их в свою очередь поместить в общедоступное для других сценариев место.

Еще один сервисный сценарий: authorize.php

Запустите редактор и создайте файл authorize.php. Для начала можно добавить в него подходящую комбинацию имени пользователя и пароля:

```
<?php

define(VALID_USERNAME, "admin");
define(VALID_PASSWORD, "super_secret");

?>
```

На этой стадии обычно создается функция, возможно, с именем authorize, get_credentials или каким-нибудь другим подходящим по смыслу. Но это ли вам нужно на самом деле? Неужели вы хотите вставлять в сценарии инструкцию require_once authorize.php, а затем непосредственно вызывать какую-нибудь функцию?

Скорее всего, вам нужно с помощью всего одной строки кода указать сценарий, который требует аутентификации:

```
require_once "../scripts/authorize.php;"
```

Затем в идеале должна пройти авторизация.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Стоит ли мириться с бесконечным количеством попыток регистрации?

Конечно же, нет. Но сейчас сценарий show_users.php допускает именно такое развитие событий: возможность все новых и новых попыток до получения приемлемого имени пользователя и пароля. По правде говоря, код примеров и схемы, которые встречаются по всему Интернету для использования в качестве стандартной аутентификации, выглядят ничуть не лучше того, что получилось у вас в сценарии show_users.php.

Конечно, есть множество способов обхода данной проблемы, но они не настолько просты, как вам бы этого хотелось. Поскольку браузер отправляет вашему сценарию несколько запросов, вам нужно найти способ определения количества запросов, сделанных вашему сценарию из... вашего сценария. А это непросто.

Способы обнаружения отправки нескольких запросов есть, и они будут изучены в следующей главе, посвященной сессиям (хотя и для более значимой цели). А сейчас нужно понять, что стандартный подход к аутентификации носит временный характер и весь этот код является всего лишь отправной точкой, а не конечным пунктом в решении этой задачи.

Стало быть, вызов функции нам не нужен. Нужен просто какой-нибудь код PHP, находящийся в основной части сценария authorize.php. Чтобы при запросе сценария authorize.php этот код запускался, проводил аутентификацию и вашему сценарию не приходилось ничего делать для получения всех преимуществ от аутентификации и авторизации.

По множеству признаков авторизация здесь похожа на наличие кода JavaScript внутри набора <script>-тегов без какой-либо функции:

```
<script type="text/javascript">
  dashboard_alert("#hits_count_dialog");
  $("#hits_count_dialog").dialog("open");
  query_results_tables();
</script>
```

Как только браузер обнаружит этот код JavaScript, он его запустит. То же самое справедливо и для кода PHP, не относящегося к функции. Поэтому вы можете поместить код авторизации непосредственно в сценарий authorize.php:

```
<?php

define(VALID_USERNAME, "admin");
define(VALID_PASSWORD, "super_secret");

if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW']) ||
    ($_SERVER['PHP_AUTH_USER'] != VALID_USERNAME) ||
    ($_SERVER['PHP_AUTH_PW'] != VALID_PASSWORD)) {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
}

?>
```

Теперь любой сценарий, имеющий инструкцию `require_once` для сценария `authorize.php`, заставит интерпретатор обработать `authorize.php`, что в свою очередь вызовет запуск кода авторизации. Тем самым будет гарантировано то, что пользователь либо зарегистрируется, либо будет принужден к регистрации. То есть мы добьемся желаемого.

Теперь удалите соответствующий код из сценария `show_users.php` и добавьте в него инструкцию `require_once` для сценария `authorize.php`:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/authorize.php';
require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';
// Теперь кода авторизации нет в этом сценарии

// Создание инструкции SELECT
$select_users =
  "SELECT user_id, first_name, last_name, email " .
  "  FROM users";

// и т. д...
?>
```

Теперь можно еще раз зайти на страницу, создаваемую сценарием `show_users.php`, и получить окно регистрации. Но это еще не все необходимые вам изменения. Аналогичную строку кода нужно добавить и в сценарий `delete_user.php`:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/authorize.php';
require_once '../scripts/database_connection.php';

// и т. д...
```

Теперь закройте браузер, чтобы были утрачены все пароли. Затем откройте браузер еще раз и перейдите непосредственно на страницу, создаваемую сценарием `delete_user.php`. Вас поприветствует окно регистрации, показанное на рис. 11.6. Самое примечательное в данном случае то, что для добавления системы защиты к другой странице понадобилась всего одна строчка кода PHP.

Рис. 11.6. Просмотр пользователей после прохождения аутентификации, которая отделена от основного содержимого ваших страниц

Но это еще не все! Если вы зарегистрировались, снова закройте браузер, а затем откройте его и перейдите на страницу сценария `show_users.php`. Как и ожидалось, вам придется зарегистрироваться. После регистрации щелкните на значке удаления одного из пользователей. Этот щелчок переместит вас на страницу сценария `delete_user.php`, и будет запущен код PHP сценария `authorize.php`. Но поскольку вы уже зарегистрировались в области, идентифицируемой как `The Social Site`, вы не получите повторного приглашения на прохождение регистрации. Если помните, область задавалась следующим кодом:

```
header('WWW-Authenticate: Basic realm="The Social Site"');
```

Любая страница, использующая такую же область, будет применять общие полномочия с другими страницами этой же области. Стало быть, после регистрации для допуска к странице `show_users.php`, имеющей такую же область, что и у страницы `delete_user.php`, ваш запрос на удаление не встретит никаких препятствий (результат показан на рис. 11.7, где нет никакого окна регистрации).

Использование общих полномочий работает только в том случае, если для данных двух страниц объявлена одна и та же область. Это еще один повод для того, чтобы убрать код аутентификации и авторизации из отдельных сценариев и поместить его в одно место, на которое ссылаются другие ваши сценарии. Ваша область будет одинакова для всех сценариев, имеющих эти ссылки.

Но остается еще одна весьма очевидная проблема...



Рис. 11.7. Окно регистрации отсутствует

Пароли не должны находиться в сценариях PHP

Не нужно забывать, что за каждым хорошим сценарием стоит большая база данных (или что-либо ей подобное). Содержание в сценарии PHP нескольких констант, определяющих допустимые имена пользователей и пароли (даже если это сервисный сценарий вроде `app_config.php` или `authorize.php`), — совершенно никудышная идея. Пароли должны располагаться исключительно в базе данных.

Получить доступ к базам данных значительно труднее, чем к сценариям, поскольку сценарии в некоторой степени доступны через Интернет. А вот базы данных удалены от обычного веб-пользователя по крайне мере на еще один уровень. Кроме того, база данных и SQL требуют знания их структуры. Сценарии являются простыми файлами, которые могут быть просмотрены. Зачастую информация в таких файлах представлена в виде простого текста. Можно и приводить подобные аргументы, но достаточно сказать только то, что база данных является более безопасным местом для паролей, чем файл `authorize.php`.

ПРИМЕЧАНИЕ

Для ограничения интернет-доступа к сценариям, особенно к тем из них, которые выполняют сервисные функции, можно предпринять различные меры. И, конечно же, можно использовать ряд неразумных решений, делающих ваши базы данных более доступными через Интернет. Но, согласно их исходному статусу, независимо от наличия надежной системы аутентификации, сценарии могут быть доступны через браузер, а простые столбцы и строки базы данных не могут.

Есть еще один серьезный повод для размещения паролей в базе данных: там уже содержится информация о пользователях! Стало быть, вы можете связать эту информацию с паролем путем добавления еще одного столбца. И как вскоре будет показано, следующим на очереди будет создание групп пользователей. Итак, прежде чем всем этим воспользоваться, нужно опять углубиться в MySQL и улучшить положение дел с аутентификацией.

Обновление таблицы users

Сначала необходимо обновить таблицу users. На данный момент у нас имеется следующее:

```
mysql> describe users;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| user_id        | int(11)      | NO   | PRI | NULL    | auto_increment |
| first_name     | varchar(20)  | NO   |     |         |                |
| last_name      | varchar(30)  | NO   |     |         |                |
| email          | varchar(50)  | NO   |     |         |                |
| facebook_url   | varchar(100) | YES  |     | NULL    |                |
| twitter_handle | varchar(20)  | YES  |     | NULL    |                |
| bio            | text         | YES  |     | NULL    |                |
| user_pic_path  | text         | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
8 rows in set (0.02 sec)
```

Все вроде бы в порядке, не хватает только имени пользователя (username) и пароля (password). Именно эти важные информационные составляющие и нужны для вашей стандартной аутентификации.

Добавьте оба данных столбца в таблицу:

```
mysql> ALTER TABLE users
    ->         ADD username VARCHAR(32) NOT NULL
    ->       AFTER user_id,
    ->         ADD password VARCHAR(16) NOT NULL
    ->       AFTER username;
```

ПРИМЕЧАНИЕ

Ключевое слово AFTER сообщает MySQL о конкретном месте добавления столбца. Использование AFTER помогает избежать ситуации , при которой важные столбцы (а username и password, конечно же, относятся именно к таким столбцам) оказались бы в конце структуры таблицы. Можно обойтись и без этого, но применение данного ключевого слова помогает улучшить организацию таблиц, особенно при использовании инструкции DESCRIBE.

Теперь нужно убедиться в том, что эти изменения действительно были внесены в таблицу:

```
mysql> describe users;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| user_id        | int(11)      | NO   | PRI | NULL    | auto_increment |
| username       | varchar(32)  | NO   |     | NULL    |                |
| password       | varchar(16)  | NO   |     | NULL    |                |
| first_name     | varchar(20)  | NO   |     |         |                |
| last_name      | varchar(30)  | NO   |     |         |                |
| email          | varchar(50)  | NO   |     |         |                |
| facebook_url   | varchar(100) | YES  |     | NULL    |                |
| twitter_handle | varchar(20)  | YES  |     | NULL    |                |
| bio            | text         | YES  |     | NULL    |                |
| user_pic_path  | text         | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
10 rows in set (0.03 sec)
```

Работа с вновь созданными недопустимыми данными

Как и ранее, при добавлении столбцов вы получили таблицу, заполненную недопустимыми строками. Поскольку и имя пользователя (username) и пароль (password) являются обязательными данными (NOT NULL) и ни одна из существующих строк не имеет значений в этих столбцах, все строки вашей таблицы нарушают данные правила.

Придать недопустимым строкам надлежащий вид можно с помощью дополнительных команд SQL. Чтобы, к примеру, обновить запись пользователя Джеймса Родея, можно использовать следующий код:

```
mysql> UPDATE users
    ->    SET username = "jroday",
    ->        password = "psych_rules"
    ->  WHERE user_id = 45;
```

После чего можно убедиться в том, что эти изменения были внесены:

```
mysql> SELECT user_id, username, password, first_name, last_name
    ->   FROM users
    ->  WHERE user_id = 45;
+---------+----------+-------------+------------+-----------+
| user_id | username | password    | first_name | last_name |
+---------+----------+-------------+------------+-----------+
|      45 | jroday   | psych_rules | James      | Roday     |
+---------+----------+-------------+------------+-----------+
1 row in set (0.00 sec)
```

Подобные изменения нужно внести в таблицу users, чтобы имена пользователей и пароли были добавлены всем вашим пользователям.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

А не подойдет ли в качестве имени пользователя адрес его электронной почты?

Похоже, что на каждом этапе жизненного пути появляется новый социальный веб-сайт, к которому вы просто не можете не присоединиться. И все большее количество подобных веб-сайтов используют в качестве ваших регистрационных данных адрес вашей электронной почты. На это есть веские причины.

- Скорее всего, всем нам проще запомнить адрес своей электронной почты, чем 50 разных имен пользователей.
- Такие адреса электронной почты, как `tommy.n@dbc.org`, намного проще читаются (и набираются), чем такие имена пользователя, как `tn1954a`.
- В базе данных приходится хранить меньший объем информации.

Тогда зачем же при такой тенденции в отношении имени пользователя создавать в таблице `users` столбец `username`? Почему бы не воспользоваться адресом электронной почты?

Во-первых, у многих людей в наше время адресов электронной почты не меньше, чем имен пользователя. Было бы, наверное, нелепо воспользоваться таким подходом, когда у человека имеется почта на Gmail, iCloud и как минимум один деловой адрес электронной почты и, возможно, один-два домена.

Во-вторых, многие не любят применять адреса электронной почты в качестве своих имен пользователя. Имя пользователя представляется чем-то более анонимным, в то время как адрес электронной почты призван привлечь кого-то к переписке. Как ни странно, многие мирятся с тем, что адрес электронной почты предоставляется при подписке на какой-нибудь информационный ресурс, но при этом не любят напоминаний о том, что в таком случае их адрес хранится на многих сайтах. Поэтому требование ввода адреса электронной почты в окно регистрации будет для таких пользователей лишним раздражителем.

И в-третьих, наверное, самое важное. Как при задействовании адреса электронной почты в качестве имени пользователя восстанавливать пользовательский пароль? Обычно в системе пользовательских имен при утрате пароля от пользователя в качестве своеобразного подтверждения требуется предоставить его имя пользователя. Когда в качестве этого имени применяется адрес электронной почты, что можно запросить в качестве подтверждения?

Притом что в применении электронного адреса в качестве имени пользователя нет ничего особенно предосудительного, лучше все же затребовать имя пользователя. Кроме того, при наличии таких фантастических программ как 1Password (`www.agilebits.com/products/1Password`), вам больше не придется испытывать трудности с управлением несколькими регистрациями. Серьезно, приобретите уже сегодня программу 1Password (хотя ее цена в $59,99 и кажется несколько завышенной) — и ваша жизнь в Интернете резко изменится.

Вам нужно получить исходные имя пользователя и пароль

А теперь нужно вернуться назад. Помните файл `create_user.html`? Он содержал довольно простую HTML-форму, которая давала исходную информацию о пользователе. Эта форма нуждается в улучшении: для начала в нее нужно добавить поля имени пользователя и пароля.

Рассмотрим существенно обновленную версию файла `create_user.html`, к которому, кроме всего прочего, добавлено поле для получения имени пользователя и два поля, в совокупности позволяющие получить пароль от новых пользователей:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
  <link href="../css/jquery.validate.password.css" rel="stylesheet"
type="text/css" />

  <script type="text/javascript" src="../js/jquery.js"></script>
  <script type="text/javascript" src="../js/jquery.validate.js"></script>
  <script type="text/javascript" src="../js/jquery.validate.password.js">
  </script>

  <script type="text/javascript">
    $(document).ready(function() {
      $("#signup_form").validate({
        rules: {
          password: {
            minlength: 6
          },
          confirm_password: {
            minlength: 6,
            equalTo: "#password"
          }
        },
        messages: {
          password: {
            minlength: "Пароль должен иметь не менее 6 символов"
          },
          confirm_password: {
            minlength: "Пароль должен иметь не менее 6 символов",
            equalTo: "Ваши пароли не совпадают."
          }
        }
      });
    });
  </script>
```

```
</head>

<body>
 <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
 <div id="example">Регистрация</div>
 <div id="content">
   <h1>Вступайте в наш виртуальный клуб</h1>
   <p>Пожалуйста, введите ниже свои данные для связи в Интернете:</p>
   <form id="signup_form" action="create_user.php"
         method="POST" enctype="multipart/form-data">
     <fieldset>
       <label for="first_name">Имя:</label>
       <input type="text" name="first_name" size="20" class="required" />
       <br />
       <label for="last_name">Фамилия:</label>
       <input type="text" name="last_name" size="20" class="required" />
       <br />
       <label for="username">Имя пользователя:</label>
       <input type="text" name="username" size="20" class="required" />
       <br />
       <label for="password">Пароль:</label>
       <input type="password" id="password" name="password"
              size="20" class="required password" />
       <div class="password-meter">
         <div class="password-meter-message"> </div>
         <div class="password-meter-bg">
           <div class="password-meter-bar"></div>
         </div>
       </div>
       <br />
       <label for="confirm_password">Подтверждение пароля:</label>
       <input type="password" id="confirm_password" name="confirm_password"
              size="20" class="required" /><br />
       <label for="email">Адрес электронной почты:</label>
       <input type="text" name="email" size="30" class="required email" />
       <br />
       <label for="facebook_url"> URL-адрес в Facebook:</label>
       <input type="text" name="facebook_url" size="50" class="url" /><br />
       <label for="twitter_handle">Идентификатор в Twitter:</label>
       <input type="text" name="twitter_handle" size="20" /><br />
       <input type="hidden" name="MAX_FILE_SIZE" value="2000000" />
       <label for="user_pic">Отправка изображения:</label>
       <input type="file" name="user_pic" size="30" /><br />
       <label for="bio">Биография:</label>
       <textarea name="bio" cols="40" rows="10"></textarea>
     </fieldset>
     <br />
     <fieldset class="center">
```

```
        <input type="submit" value="Вступить в клуб" />
        <input type="reset" value="Очистить и начать все с начала" />
      </fieldset>
    </form>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

Кроме двух новых полей в эту версию формы добавлена библиотека jQuery, доступная на сайте www.jquery.com. jQuery является бесплатной загружаемой библиотекой, которая делает почти все в JavaScript намного проще. Вдобавок к ядру библиотеки jQuery здесь используются два дополнительных модуля jQuery: один для общей проверки, а второй исключительно для проверки пароля. Вы можете загрузить оба этих дополнительных модуля с сайта www.jquery.bassistance.de.

Сохраните обновленную версию файла `create_user.html` и проверьте ее в работе. Исходная страница имеет точно такой же вид, но теперь у вас есть проверка большинства полей формы и удобный индикатор надежности пароля (рис. 11.8).

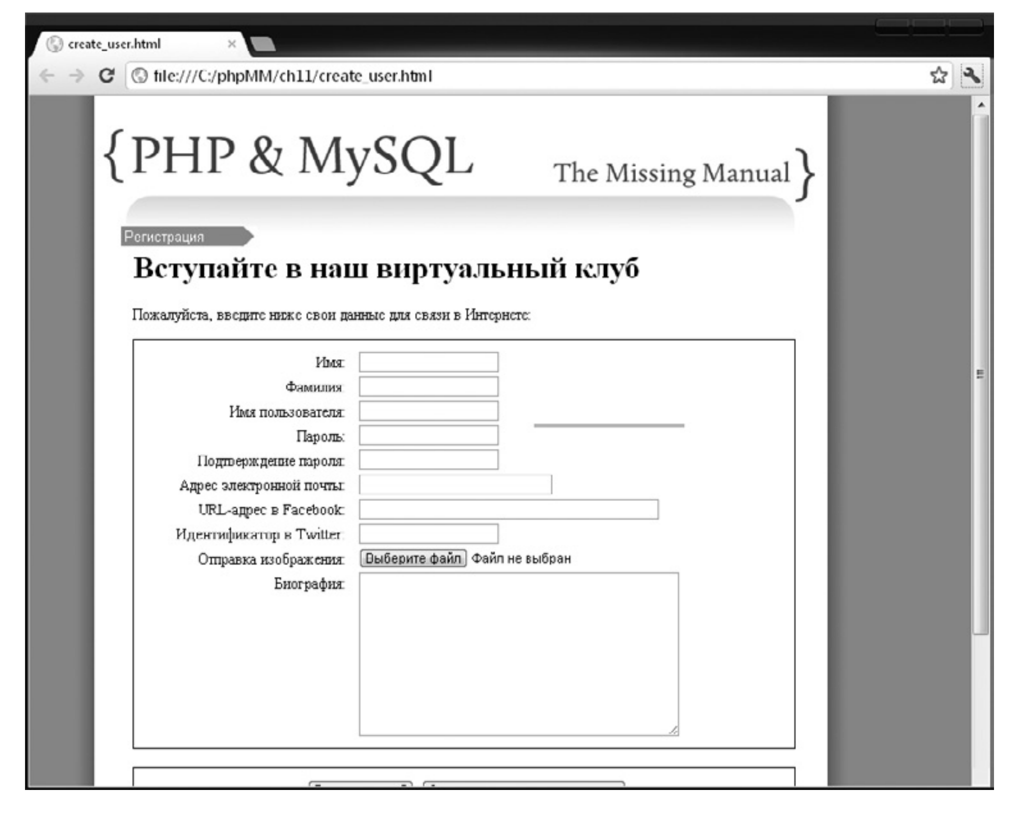


Рис. 11.8. Исходная форма с новыми полями и индикатором надежности пароля

Итак, новая версия `create_user.html` выглядит практически так же, как и прежняя. К форме добавился индикатор надежности пароля. Хотя пока пользователь не станет вводить пароль, индикатор себя не проявит. Самое главное, что к этой форме добавлены поля имени пользователя и два поля для ввода пароля: начального ввода и подтверждения. Чтобы скрыть символы, вводимые пользователем, атрибут `type` этих полей должен иметь значение `password`.

Библиотека jQuery и дополнительный модуль jQuery, предназначенный для проверки приемлемости данных, превращают проверку в пару пустяков (рис. 11.9). Затратив минимальные усилия, вы получаете проверку соответствия типов, проверку длины, произвольно настроенные сообщения об ошибках и многое другое. Вы можете проверять адреса электронной почты и телефонные номера. И это при довольно быстрой загрузке и нескольких строках кода JavaScript!

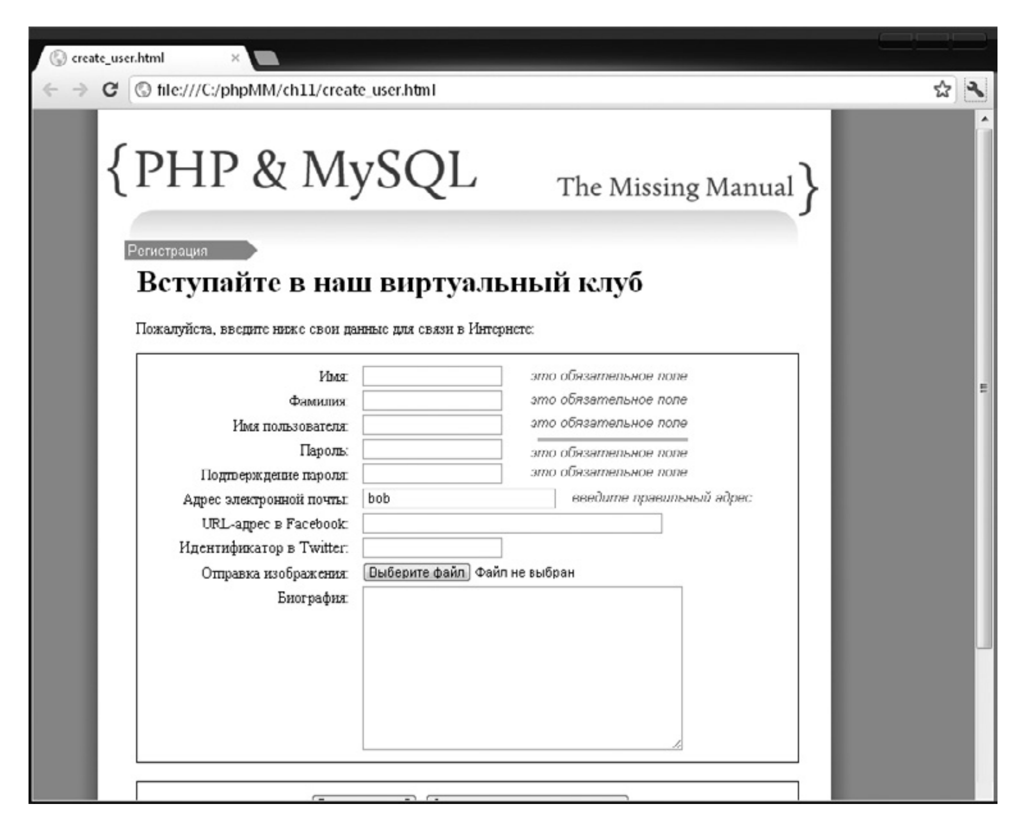


Рис. 11.9. Проверка вводимых данных

Модуль проверки приемлемости пароля является дополнением к jQuery-модулю проверки (рис. 11.10). Он добавляет индикатор надежности, который требует ввода стойких паролей. Это весьма полезное свойство, причем оно никоим образом

не увеличивает рабочую нагрузку. Все это вы получаете просто так, еще до того как ваши данные попадут к вашим PHP-сценариям.

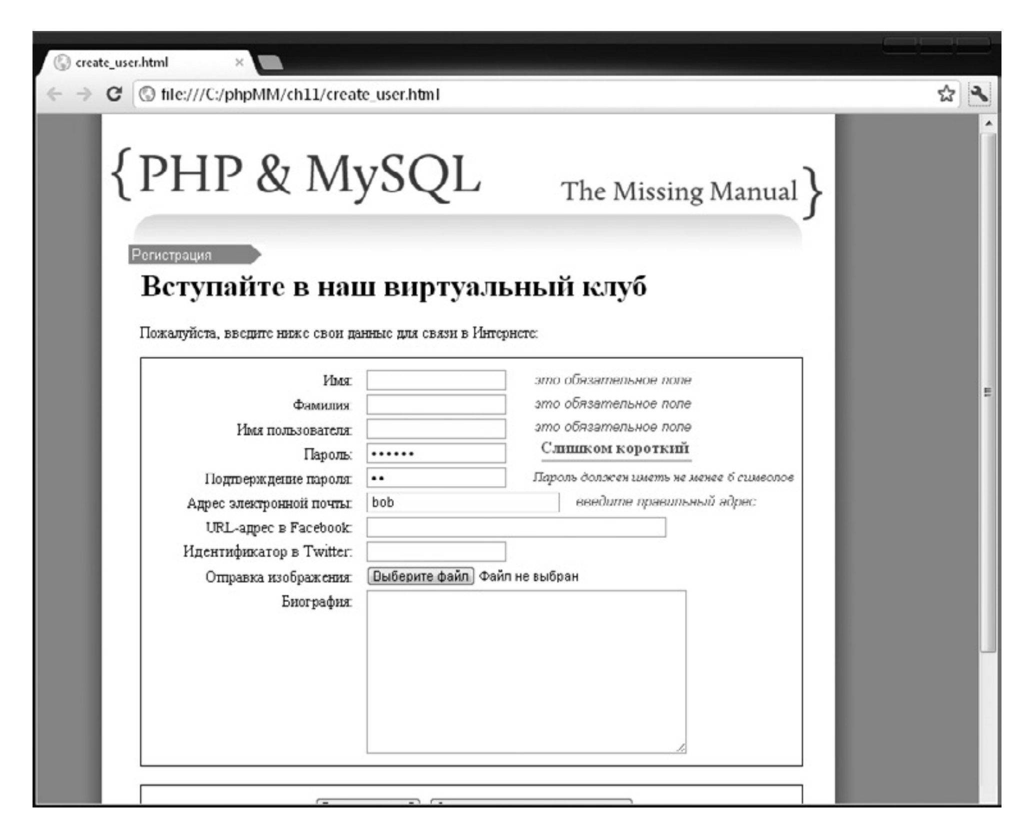


Рис. 11.10. Индикатор надежности пароля

Теперь вы получаете от пользователей правильную информацию. Настало время обновить код PHP, чтобы он мог со всем этим работать.

Вставка имени пользователя и пароля

Теперь можно также обновить сценарий `create_user.php`. Это довольно простое обновление, не требующее особых усилий. Хотя результат этого обновления будет весьма впечатляющим:

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/database_connection.php';

$upload_dir = SITE_ROOT . "uploads/profile_pics/";
```

```
$image_fieldname = "user_pic";
// Потенциальные PHP-ошибки отправки файлов
$php_errors = array(1 => 'Превышен макс. размер файла, указанный в php.ini',
                2 => 'Превышен макс. размер файла, указанный в форме HTML',
                3 => 'Была отправлена только часть файла',
                4 => 'Файл для отправки не был выбран.');

$first_name = trim($_REQUEST['first_name']);
$last_name = trim($_REQUEST['last_name']);
$username = trim($_REQUEST['username']);
$password = trim($_REQUEST['password']);
$email = trim($_REQUEST['email']);
$bio = trim($_REQUEST['bio']);
$facebook_url = str_replace("facebook.org", "facebook.com", trim($_
REQUEST['facebook_url']));
$position = strpos($facebook_url, "facebook.com");
if ($position === false) {
  $facebook_url = "http://www.facebook.com/" . $facebook_url;
}

$twitter_handle = trim($_REQUEST['twitter_handle']);
$twitter_url = "http://www.twitter.com/";
$position = strpos($twitter_handle, "@");
if ($position === false) {
  $twitter_url = $twitter_url . $twitter_handle;
} else {
  $twitter_url = $twitter_url . substr($twitter_handle, $position + 1);
}

// Проверка отсутствия ошибки при отправке изображения
($_FILES[$image_fieldname]['error'] == 0)
  or handle_error("сервер не может получить выбранное вами изображение.",
                  $php_errors[$_FILES[$image_fieldname]['error']]);

// Является ли этот файл результатом нормальной отправки?
@is_uploaded_file($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы попытались совершить безнравственный поступок. Позор!",
                  "Запрос на отправку: файл назывался " .
 ['tmp_name']}'");

// Действительно ли это изображение?
@getimagesize($_FILES[$image_fieldname]['tmp_name'])
  or handle_error("вы выбрали файл для своего фото, " .
                  "который не является изображением.",
                  "{$_FILES[$image_fieldname]['tmp_name']} " .
                  "не является файлом изображения.");
// Присваивание файлу уникального имени
$now = time();
while (file_exists($upload_filename = $upload_dir . $now .
                                     '-' .
                                     $_FILES[$image_fieldname]['name'])) {
```

```
    $now++;
}

// И наконец, перемещение файла на его постоянное место
@move_uploaded_file($_FILES[$image_fieldname]['tmp_name'],
                    $upload_filename)
  or handle_error("возникла проблема сохранения вашего изображения " .
                  "в его постоянном месте.",
                  "ошибка, связанная с правами доступа при перемещении " .
                  "файла в {$upload_filename}");

$insert_sql = sprintf("INSERT INTO users " .
                             "(first_name, last_name, username, " .
                              "password, email, " .
                              "bio, facebook_url, twitter_handle, " .
                              "user_pic_path) " .
                      "VALUES ('%s', '%s', '%s', '%s', '%s',
                               '%s', '%s', '%s', '%s');",
                      mysql_real_escape_string($first_name),
                      mysql_real_escape_string($last_name),
                      mysql_real_escape_string($username),
                      mysql_real_escape_string($password),
                      mysql_real_escape_string($email),
                      mysql_real_escape_string($bio),
                      mysql_real_escape_string($facebook_url),
                      mysql_real_escape_string($twitter_handle),
                      mysql_real_escape_string($upload_filename));

// Вставка пользователя в базу данных
mysql_query($insert_sql);

// Перенаправление пользователя на страницу, показывающую информацию
// о пользователе
header("Location: show_user.php?user_id=" . mysql_insert_id());
?>
```

ПРИМЕЧАНИЕ

Даже притом, что изменились всего лишь несколько строк, появилась хорошая возможность проверить свою текущую версию сценария create_user.php (вместе с create_user.html). Убедитесь в том, что эти файлы включают все текущие изменения, внесенные в главах 9 и 10, которые касались обработки изображения. Если вы видите, что код безнадежно отстал, всегда есть возможность снова загрузить нужные сценарии с веб-страниц, посвященной серии недостающих руководств.

Как обычно, попробуйте ввести какие-нибудь примерные данные и убедитесь в том, что появилось примерно такое же изображение, как на рис. 11.11. Это и будет подтверждением работоспособности всего, что было сделано. Кроме того, убедитесь, что к списку востребованных с помощью инструкции `require_once` сценариев не добавлен сценарий `authorize.php`. Вряд ли вы потребуете от пользователей зарегистрироваться для работы с формой, с помощью которой они сообщают вашему приложению о своих имени пользователя и пароле.



Рис. 11.11. Запретительное окно регистрации

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

А не должен ли сценарий create_user.php проверять запрошенное имя пользователя?

Да, должен. То, что в текущей версии create_user.php пользователи вводят данные в базу без проверки имени пользователя на уникальность, создает большую проблему. Конечно, можно все решить на уровне базы данных, но при этом вы получите всего лишь неприятную ошибку.

В самом простом случае вы можете воспользоваться запросом SELECT по желаемому имени. И если будет возвращен кто-нибудь из пользователей, перенаправить пользователя на страницу ошибки, используя функцию handle_error. Но это слишком примитивное решение. Оно полностью прервет ход выполнения программы, и пользователь, если он еще не получил от вашего приложения всего, что хотел, будет вынужден снова вводить всю свою информацию в форму регистрации.

Лучше превратить create_user.html в сценарий или даже вписать его содержимое в текущую версию сценария create_user.php. В любом случае, если имя пользователя уже кем-то применяется, пользователь должен быть перенаправлен на форму регистрации, заполненную всей его предыдущей информацией, и сообщение должно подсказать ему, что нужно выбрать другое имя пользователя. Если вам захочется углубиться в решение данной задачи, сделайте все, о чем сказано раньше, но только с помощью Ajax, чтобы страница регистрации никогда не подвергалась перезагрузке.

Итак, где же код для создания этого сообщения? Все в вашей голове и в ваших руках. На этой стадии вашего путешествия в мир PHP вы уже готовы самостоятельно справляться со все более сложными проблемами вроде этой. В качестве источников новых технологий, которые касаются, к примеру, аутентификации, описанной в данной главе, или сессий, рассматриваемых в главе 12, можно воспользоваться ресурсами Интернета. Вы уже вполне способны самостоятельно разрабатывать новые варианты применения изученных вами технологий.

Подключение сценария authorize.php к таблице users

На данный момент осталось закрыть только одну зияющую дыру: подключить сценарий authorize.php. Сейчас в нем допускаются только одно имя пользователя и пароль, заданные довольно примитивными константами:

```
define(VALID_USERNAME, "admin");
define(VALID_PASSWORD, "super_secret");
```

Но теперь для сценария authorize.php есть таблица users, из которой можно извлекать имена пользователей и пароли.

К счастью, подключение сценария authorize.php к таблице users не представляет собой какую-либо слишком трудную задачу. Фактически внесение правок в authorize.php заключается всего лишь в связывании вместе того, чем вы уже занимались. Нужно удалить эти две константы и добавить инструкцию require_once, чтобы запросить сценарий database_connection.php, который понадобится для взаимодействия с таблицей users:

```
<?php

require_once 'database_connection.php';

// define(VALID_USERNAME, "admin");         ЭТУ СТРОКУ НУЖНО УДАЛИТЬ
// define(VALID_PASSWORD, "super_secret");  ЭТУ СТРОКУ НУЖНО УДАЛИТЬ

if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW']) ||
    ($_SERVER['PHP_AUTH_USER'] != VALID_USERNAME) ||
    ($_SERVER['PHP_AUTH_PW'] != VALID_PASSWORD)) {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
}

?>
```

Теперь необходимо немного урезать содержимое слишком большой инструкции if. Ее первая часть будет работать по-прежнему. Если сверхглобальная переменная $_SERVER не имеет значений для PHP_AUTH_USER или PHP_AUTH_PW, то заголовки должны быть отправлены браузеру, заставляя его вывести окно регистрации.

Но теперь у нас уже нет констант VALID_USERNAME или VALID_PASSWORD, с которыми должны сравниваться значения, введенные пользователем. Поэтому часть инструкции if должна быть убрана. Должен остаться следующий код:

```
if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW'])) {
  header('HTTP/1.1 401 Unauthorized');
```

```
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
}
```

ПРИМЕЧАНИЕ

Все, что здесь находится после инструкции if (если), по сути является инструкцией else (или же), даже если само ключевое слово else отсутствует. Если тело инструкции if выполняется, будет вызвана функция exit, завершающая работу сценария. Поэтому остальная часть сценария будут выполнена только в том случае, если значения с ключами PHP_AUTH_USER и PHP_AUTH_PW будут присутствовать в массиве $_SERVER.

Далее сценарию нужно получить то, что было введено пользователем. Если сценарий доберется до этого места, значит, пользователь что-то ввел в окно и нужно сравнить введенную пользователем информацию со значениями в базе данных. Но этим вы уже занимались несколько раз. Для этого понадобятся дополнительные вызовы функций sprintf и mysql_real_escape_string, обе из которых вы уже неоднократно использовали:

```
<?php

require_once 'database_connection.php';

if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW'])) {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
 exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
}

// Поиск предоставленных пользователем полномочий
$query = sprintf("SELECT user_id, username FROM users " .
                 " WHERE username = '%s' AND " .
                 "       password = '%s';",
            mysql_real_escape_string(trim($_SERVER['PHP_AUTH_USER'])),
            mysql_real_escape_string(trim($_SERVER['PHP_AUTH_PW'])));

$results = mysql_query($query);

?>
```

Здесь нет ничего нового. И вы знаете, как можно получить результаты. Но на этот раз помимо волнений за результаты ответа встает более серьезный вопрос их наличия в принципе. Если предоставлена строка, соответствующая введенным имени пользователя и паролю, стало быть, это законный пользователь. (Или он по крайней мере присвоил чьи-то полномочия. Но «присвоение» здесь вполне допускается.)

Следовательно, сначала нужно посмотреть, есть ли какие-нибудь результаты. Если нет, сценарий должен оказаться там, где бы он был в предыдущей версии, когда имя пользователя и пароль считались неподходящими. А это означает повторную отправку тех же самых заголовков:

```
if (mysql_num_rows($results) == 1) {
  // Все в порядке! Этого пользователя можно пропустить дальше.
} else {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");

}
```

ПРИМЕЧАНИЕ

Если вас смущает, что этот код, отправляющий заголовки, идентичен коду, который ранее уже встречается в сценарии, переместитесь в начало сценария, создайте функцию, которая будет выводить заголовки и принимать сообщение, передаваемое функции exit, а затем дважды вызовите эту функцию в сценарии authorize.php.

Теперь осталось только кое-что уточнить. Поскольку пользователь уже зарегистрировался, нужно дать любому сценарию, вызывающему `authorize.php`, доступ к данным только что зарегистрировавшегося пользователя:

```
if (mysql_num_rows($results) == 1) {
  $result = mysql_fetch_array($results);
  $current_user_id = $result['user_id'];
  $current_username = $result['username'];
} else {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");

}
```

Теперь весь сценарий, обновленный и, конечно же, улучшенный, имеет следующий вид:

```
<?php

require_once 'database_connection.php';

if (!isset($_SERVER['PHP_AUTH_USER']) ||
    !isset($_SERVER['PHP_AUTH_PW'])) {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
```

```
       "Проходите дальше. Здесь вам нечего смотреть.");

}

// Поиск предоставленных пользователем полномочий
$query = sprintf("SELECT user_id, username FROM users " .
                 " WHERE username = '%s' AND " .
                 "       password = '%s';",
            mysql_real_escape_string(trim($_SERVER['PHP_AUTH_USER'])),
            mysql_real_escape_string(trim($_SERVER['PHP_AUTH_PW'])));

$results = mysql_query($query);

if (mysql_num_rows($results) == 1) {
  $result = mysql_fetch_array($results);
  $current_user_id = $result['user_id'];
  $current_username = $result['username'];
} else {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");

}

?>
```

Проверьте сценарий в работе. Создайте пользователя (или добавьте имя и пароль для пользователя, уже существующего в вашей базе данных), а затем закройте и снова откройте браузер для переустановки любых сохраненных полномочий. Перейдите на страницу, выводимую сценарием show_users.php, или на любую другую страницу, которая запрашивает сценарий authorize.php. Перед вами должно появиться окно регистрации, дающее возможность ввести в него такие же значения, которые хранятся в базе данных, и увидеть запрошенную страницу.

Пароли обеспечивают безопасность, но и сами они должны быть защищены

Ваше новое средство регистрации, работающее с данными, которые хранятся в базе, открывает новые возможности. Прежде всего вы можете создать в базе данных группы и позволить пользователям получать доступ к определенным частям своего приложения на основе их принадлежности к той или иной группе. Поэтому вместо допуска практически любого пользователя к show_users.php нужно допустить к странице, создаваемой этим сценарием, только пользователей, входящих в группу admin.

Но перед тем как все это сделать, взгляните еще раз на ваши недавние инструкции SQL и результаты:

```
mysql> SELECT user_id, username, password, first_name, last_name
    ->   FROM users
    ->  WHERE user_id = 45;
+---------+----------+-------------+------------+-----------+
| user_id | username | password    | first_name | last_name |
+---------+----------+-------------+------------+-----------+
|      45 | jroday   | psych_rules | James      | Roday     |
+---------+----------+-------------+------------+-----------+
1 row in set (0.00 sec)
```

Нет ли здесь чего-нибудь странного? (Все в порядке, кроме пароля, который выбрал James Roday. Конечно, Psych — оригинальный выбор, но его нельзя отнести к тем паролям, которые нельзя взломать.)

Более очевидной проблемой является присутствие пароля в базе данных в виде простого текста. И даже если вы новичок в мире аутентификации и авторизации, вам, наверное, приходилось сталкиваться с таким понятием, как шифрование. При шифровании берется фрагмент секретной информации и превращается в нечто нечитаемое простыми смертными. Замысел заключается в том, чтобы никто другой, кроме владельца пароля, даже умудренный опытом и обладающий обширными знаниями программист, не мог увидеть пользовательский пароль в виде обычного текста.

Следовательно, необходимо получить средство шифрования пароля для получения чего-то нечитаемого. И вы уже знаете, что будет дальше: в PHP есть функция, предназначенная для решения данной задачи.

Шифрование текста с помощью функции crypt

Сначала нужно превратить пароль во что-нибудь нечитаемое. Это можно сделать с помощью PHP-функции crypt. Ей передается строка (и необязательный второй аргумент, который вам скоро понадобится), и она выдает результат, больше похожий на тарабарщину:

```
$encrypted_password = crypt($password);
```

Чтобы посмотреть эту функцию в работе, внесите в сценарий create_user.php следующие изменения:

```
$insert_sql = sprintf("INSERT INTO users " .
                             "(first_name, last_name, username, " .
                              "password, email, " .
                              "bio, facebook_url, twitter_handle, " .
                              "user_pic_path) " .
                      "VALUES ('%s', '%s', '%s', '%s', '%s',
                               '%s', '%s', '%s', '%s');",
```

```
                  mysql_real_escape_string($first_name),
                  mysql_real_escape_string($last_name),
                  mysql_real_escape_string($username),
                  mysql_real_escape_string(crypt($password)),
                  mysql_real_escape_string($email),
                  mysql_real_escape_string($bio),
                  mysql_real_escape_string($facebook_url),
                  mysql_real_escape_string($twitter_handle),
                  mysql_real_escape_string($upload_filename));
```

Теперь создайте нового пользователя, позвольте сценарию create_user.php сохранить его данные, а затем проверьте наличие этих данных в таблице users:

```
mysql> SELECT user_id, username, password, last_name
    ->   FROM users
    ->  WHERE user_id = 51;
+---------+----------+-----------------+----------+
| user_id | username | password        | last_name |
+---------+----------+-----------------+----------+
|      51 | traugott | $1$qzifqLu4$0C88 | Traugott  |
+---------+----------+-----------------+----------+
1 row in set (0.00 sec)
```

Теперь уже значительно лучше. Вообще-то, вам, наверное, нужно увеличить размер поля пароля, поскольку функция crypt увеличивает длину исходного введенного пароля.

```
ALTER TABLE users
     CHANGE password
            password VARCHAR(50) NOT NULL;
```

ПРИМЕЧАНИЕ

Имя поля password здесь применено дважды неспроста. Когда задаются изменения столбца, сначала указывается исходное имя этого столбца. Затем дается новое имя столбца, новый тип столбца и любые изменения (такие как NOT NULL). В данном примере, поскольку исходное и новое имена совпадают, вы просто дважды указываете имя password.

Теперь пароль попадает в базу данных, а как насчет извлечения его оттуда?

Однонаправленное шифрование с помощью функции crypt

Функция crypt по определению осуществляет однонаправленное шифрование. Это означает, что как только пароль будет зашифрован, он не может быть расшифрован. Это проблема для вас как программиста, но хорошее подспорье для пользователей. Это означает, что даже администраторы приложений, с которыми работают пользователи, не могут пробраться в свою базу данных и извлечь из нее пароли пользователей.

При подобной попытке они получат только зашифрованную версию. И не существует какой-то особой формулы или магической команды, позволяющей им

получить оригиналы этих паролей. Тем самым защищены и пользователи, и вы сами. Если вы не сможете получить незашифрованный пароль, то вы не сможете быть обвиненным, к примеру, в каком-то мошенничестве.

Но если вы не можете расшифровать значение пароля в базе данных, то как вы поймете, что пользователь ввел правильный пароль?

Очень просто: вы можете зашифровать предоставленный им пароль и сравнить это зашифрованное значение с зашифрованным значением в базе данных. Если эти зашифрованные значения совпадут, значит, все хорошо и вам не нужно знать истинное значение пароля.

Итак, вам нужно в сценарии authorize.php (там, где проверяются пароли) иметь что-нибудь вроде следующего кода:

```
// Поиск предоставленных пользователем полномочий
$query = sprintf("SELECT user_id, username FROM users " .
                 " WHERE username = '%s' AND " .
                 "       password = '%s';",
            mysql_real_escape_string(trim($_SERVER['PHP_AUTH_USER'])),
            mysql_real_escape_string(
              crypt(trim($_SERVER['PHP_AUTH_PW']))));
```

ВНИМАНИЕ

Уделите достаточное внимание всем этим закрывающим круглым скобкам. В них очень легко запутаться, и вам вряд ли захочется столкнуться с трудно обнаруживаемой вложенной ошибкой, вызванной пропуском одной круглой скобки.

Теперь все нужно испытать в работе. При создании пользователя происходит шифрование пароля, а затем выполняется шифрование значения, сравниваемого с паролем при регистрации пользователя.

К сожалению, на данной стадии испытания закончатся тем, что вы застрянете на картинке, показанной на рис. 11.11. Есть одно упущение, которое относится к внутренней работе функции crypt.

Так что же случилось?

При шифровании используется соль

Помните о кратко упомянутом втором аргументе функции crypt? Его называют солью. *Соль* — это ключ, составленный обычно из нескольких символов, который применяется в генерации однонаправленного шифрования, используемого такими функциями, как crypt. Соль помогает обеспечить случайный характер и безопасность пароля.

До сих пор, не предоставляя функции соль, вы позволяли функции crypt определять какую-то соль самостоятельно. Но пока двум разным вызовам функции crypt не будет предоставлена одна и та же соль, результаты шифрования совпадать не будут. Иными словами, вызов функции crypt в отношении одной и той же строки дважды без предоставления соли приведет к получению двух разных результатов.

Создайте простой сценарий под названием test_salt.php:

```
<?php

$input = "secret_string";
$first_output = crypt($input);
$second_output = crypt($input);

echo "Первый вывод: {$first_output}\n\n";
echo "Второй вывод: {$second_output}\n\n";

?>
```

Теперь запустите этот сценарий в терминале командной строки:

```
yellowta@yellowtagmedia.com [~/www/phpMM/ch11]# php test_salt.php
X-Powered-By: PHP/5.2.17
Content-type: text/html

Первый вывод: $1$ciU1qEcc$XFT9G7FD/4K/L1Kl.bd.q/

Второй вывод: $1$7cLtF/bc$Js6rEk5RHg4PujAkVOOSG1
```

Но это еще не все. Запустите данный сценарий еще раз, и вы получите два совершенно других результата.

Но при внесении одного изменения все возвращается в нужное русло:

```
<?php

$input = "secret_string";
$salt = "salt";
$first_output = crypt($input, $salt);
$second_output = crypt($input, $salt);

echo "Первый вывод: {$first_output}\n\n";
echo "Второй вывод: {$second_output}\n\n";

?>
```

Теперь запустите эту обновленную версию и радуйтесь результатам:

```
yellowta@yellowtagmedia.com [~/www/phpMM/ch11]# php test_salt.php
X-Powered-By: PHP/5.2.17
Content-type: text/html

Первый вывод: sazmIw2D3KJ/M

Второй вывод: sazmIw2D3KJ/M
```

Итак, вы должны обеспечить при обоих вызовах функции crypt в сценариях вашего приложения использование одной и той же соли. Теперь вы можете просто создать новую константу, но есть более подходящее решение: использовать в качестве соли само имя пользователя! Вы можете полностью лишиться своих сценари-

ев и любых констант, определяющих соль, а ваша аутентификация будет работать по-прежнему.

Имя пользователя всегда находится рядом с паролем, стало быть, вы, по сути, обеспечиваете, что имя пользователя и пароль являются реальной объединенной комбинацией.

Сначала обновите сценарий create_user.php (да, еще раз!), чтобы использовать в качестве соли предоставленное пользователем имя:

```
$insert_sql = sprintf("INSERT INTO users " .
                       "(first_name, last_name, username, " .
                        "password, email, " .
                        "bio, facebook_url, twitter_handle, " .
                        "user_pic_path) " .
                 "VALUES ('%s', '%s', '%s', '%s', '%s',
                          '%s', '%s', '%s', '%s');",
                 mysql_real_escape_string($first_name),
                 mysql_real_escape_string($last_name),
                 mysql_real_escape_string($username),
                 mysql_real_escape_string(crypt($password, $username)),
                 mysql_real_escape_string($email),
                 mysql_real_escape_string($bio),
                 mysql_real_escape_string($facebook_url),
                 mysql_real_escape_string($twitter_handle),
                 mysql_real_escape_string($upload_filename));
```

Теперь внесите точно такие же изменения в сценарий authorize.php. Следует напомнить, что в этом сценарии имя пользователя берется из сверхглобальной переменной $_SERVER:

```
// Поиск предоставленных пользователем полномочий
$query = sprintf("SELECT user_id, username FROM users " .
                 " WHERE username = '%s' AND " .
                 "       password = '%s';",
         mysql_real_escape_string(trim($_SERVER['PHP_AUTH_USER'])),
         mysql_real_escape_string(
           crypt(trim($_SERVER['PHP_AUTH_PW']),
                 $_SERVER['PHP_AUTH_USER'])));
```

А теперь, наконец, создайте нового пользователя (надеюсь, список ваших друзей еще не исчерпан!). Затем попробуйте зарегистрироваться, используя эти имя пользователя и пароль.

Появление того же самого экрана сценария show_users.php имеет куда более существенное значение, чем предоставление простой возможности удаления пользователей. Оно означает, что вы получили надежную, работоспособную систему аутентификации. Мои поздравления. Радуйтесь этому, но... нужно решить еще одну серьезную проблему.

12 Cookie-файлы, вопросы регистрации и избавление от примитивных окон

Уже пройден путь от наблюдения за кодом PHP как за неким странным, зашифрованным набором символов в угловых скобках со знаком доллара и до создания собственного приложения, включая интеграцию с базой данных MySQL, аутентификацию, перенаправление и довольно приличный набор сервисных функций. Но вы все же пока, наверное, не собираетесь продавать свое мини-приложение, в котором регистрируются идентификатор в Twitter и URL-адрес в Facebook компании Google за миллиард долларов. Но зато вы получили довольно неплохое представление о том, как нужно думать категориями PHP и как создавать структуру сценариев для решение тех или иных проблем.

Но перед тем как подстроить это приложение к вашим собственным целям, остались проблемы, с которыми необходимо справиться. Вот только некоторые вопросы, которые нужно решить.

- Нужен более качественный экран регистрации. Примитивное серое окно не нравится никому, пользователям хочется фирменной, стильной формы регистрации.
- Необходимо более четкое уведомление о том, зарегистрировался пользователь в приложении или нет.
- Нужен способ отмены регистрации.
- Нужны два уровня аутентификации: один, чтобы попасть в основную часть приложения, и еще один административный уровень, чтобы попасть на такие страницы, как `show_users.php` и `delete_user.php`.

- Нужна навигация, которая будет меняться на основе регистрации пользователя и той группы, к которой он принадлежит.

Почти все в этом списке касается регистрации при входе в приложение, и это совсем не случайно. Красивый экран или возможность сгруппировать пользователей — на все это при разработке системы аутентификации и авторизации будет затрачено ничуть не меньше времени, чем на разработку других систем вашего приложения. Даже если у вас уже имеется какой-нибудь готовый код для получения имени пользователя и пароля, большинство веб-приложений создано с использованием компонентов, из которых вам могли бы подойти только некоторые. Кроме того, веб-приложение на основе регистрации пользователей показывает им разные вещи и предоставляет различные функциональные возможности.

Нужно понять, как хранить полномочия пользователей, как перенаправлять их на разные страницы вашего сайта и как решать вопросы, в основе которых лежит следование информации, предоставленной пользователем.

Выход за рамки стандартной аутентификации

На данный момент ваша аутентификация использует встроенные в браузер возможности, связанные с применением HTTP-протокола. К сожалению, при всей пользе от HTTP-аутентификации она дает вам весьма скудное визуальное представление, печальным напоминанием которого может послужить изображение, показанное на рис. 12.1. В действительности основная проблема этой страницы регистрации заключается не в ее неприглядном внешнем виде, а в том, что она не дает столько средств управления, сколько вам в конечном счете понадобится. В случае неудачной регистрации пользователя вы не можете предоставить ему свое сообщение. Вы должны заставить пользователя запросить страницу, которая выдаст заголовки. И в конце концов, этот способ дает слишком мало возможностей управления.

Рис. 12.1. Эта страница регистрации предоставляет слишком мало средств управления

А теперь запомните следующее: не считая функции начальной регистрации или показа стандартной главной страницы, это начало практически всего, что есть в вашем приложении. Следовательно, любая работа, проделанная с прекрасным дизайнером, любые красивые настройки CSS и цветовые схемы или любой искусный код HTML5 и SVG будут напрасны, потому что все это будет скрадываться этим раздражающим серым окном. Хуже того, когда пользователь не входит в приложение, перед его глазами снова и снова появляется это надоедливое окно.

Но изменить ситуацию с помощью одной какой-нибудь одной правки не удастся. Понадобится полная переработка способа, с помощью которого пользователи получают доступ к вашему сайту.

Начало с создания стартовой страницы

Любой сайт, требующий регистрации, должен предоставить вам что-нибудь, куда можно приземлиться перед тем, как попасть на страницу регистрации. Поэтому вам нужно что-нибудь простое и эффективное в качестве стартового места для пользователей вашего сайта. В нем они должны получить возможность зарегистрироваться или создать новую учетную запись.

Рассмотрим простую версию именно такой страницы. Назовите файл index.html, чтобы создаваемая им страница со временем могла стать вашей стартовой страницей по умолчанию:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="content">
    <div id="home_banner"></div>
    <div id="signup">
      <a href="create_user.html"><img src="../images/sign_me_up.png" /></a>
      <a href="signin.html"><img src="../images/sign_me_in.png" /></a>
    </div>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

Как выглядит эта страница, можно увидеть на рис. 12.2. Вам, наверное, не нужно отравлять этот сайт на конкурс дизайна, созданного по технологии Веб 2.0. И все же он дает вам отправную точку: вы требуете от пользователя либо провести на-

чальную регистрацию, либо зарегистрироваться для входа на сайт. Страница начальной регистрации у вас уже есть, а как насчет регистрации для входа на сайт? Похоже, здесь понадобится новая страница, а то и две, а также код PHP и избавление от ранее использовавшегося окна регистрации, выводимого HTTP-аутентификацией.



Рис. 12.2. Образец стартовой страницы

Провести первичную регистрацию пользователей на сайте довольно легко: нужно просто указать их данные в форме сценария `create_user.html` и получить эффект от уже проделанной работы. А вот ссылка на файл `signin.html` вызывает новые вопросы, на которые нужно дать ответ. Главный вопрос: что именно должно произойти при регистрации пользователя для входа в приложение?

Управление регистрацией пользователя при входе в приложение

Здесь, безусловно, нужна форма, в которую пользователь может ввести информацию. Далее события должны развиваться таким образом, чтобы форма отправляла данные сценарию, который займется проверкой имени пользователя и пароля. Это уже отличается от того, что есть: на данный момент аутентификация происходит в качестве побочного эффекта от запроса страницы, требующей запуска сценария authorize.php. Поэтому считать это явной формой регистрации нельзя, но теперь вам нужна именно такая форма.

Затем сценарий, получающий информацию из формы регистрации для входа в приложение должен проверить полномочия пользователя. Это не составляет труда, такую задачу уже выполняет сценарий authorize.php, и даже притом, что он использует переменную $_SERVER, нетрудно изменить получение введенной информации из формы регистрации. Но здесь есть еще один трудный вопрос: если полномочия не подтверждаются, то форму регистрации нужно показать еще раз, и предпочтительнее с изначально введенным именем пользователя или как минимум с сообщением о том, что при регистрации произошла ошибка.

ПРИМЕЧАНИЕ

Нет ничего более разочаровывающего, чем такая форма регистрации, которая выдает вам только пустое место, никогда не сообщая о том, что она получила ваши полномочия и они были отклонены. Обратная связь с пользователем является весьма важной стороной в любой системе регистрации.

Основные направления разработки должны быть следующими.

1. Регистрационная форма (HTML) принимает введенные пользователем имя пользователя и пароль и отправляет их в адрес следующего средства.
2. Сценарий аутентификации (PHP) проверяет наличие такого имени пользователя и пароля в базе данных. Если они там найдены, переводит пользователя на защищенную страницу, такую как страница пользовательского профиля (show_user.php), и оповещает его о том, что он произвел зарегистрированный вход в систему. Если его полномочия не подтверждаются, то он переводится назад на регистрационную форму (HTML).

И тут возникает проблема: как может HTML-форма отображать сообщение об ошибке при определенном условии? Или как она может предварительно заполнять поле имени пользователя?

Наличие такой формы регистрации в формате HTML накладывает серьезные ограничения не на начальное отображение, а на ситуацию, при которой регистрация проходит неудачно. Тогда вам требуется код PHP на вашей стороне.

Вполне очевидным решением является преобразование страницы регистрации в сценарий PHP. И в конечном итоге вы получаете такую последовательность средств.

1. Регистрационная форма (на этот раз на PHP) принимает введенные пользователем имя пользователя и пароль. Отправляет их в адрес сценария аутентификации (PHP).
2. Сценарий аутентификации (PHP) проверяет наличие такого имени пользователя и пароля в базе данных. Если они там найдены, переводит пользователя на защищенную страницу, такую как страница пользовательского профиля (show_user.php), и оповещает его о том, что он произвел зарегистрированный вход в систему. Если его полномочия не подтверждаются, то он переводится назад на регистрационную форму (PHP).
3. Регистрационная форма (PHP) теперь может отображать пользовательскую ошибку и перезагружать введенное имя пользователя.

Но почему не пойти еще дальше? Все это относится к шагам в процессе регистрации. Так почему бы вместо двух сценариев не иметь один, отправляющий данные себе самому и либо перенаправляющий пользователя при удачной регистрации, либо заново самостоятельно показывающий собственную информацию при неудачной регистрации? (Если идея сценария, отправляющего данные себе самому, звучит несколько странно, прочитайте следующую врезку «Код PHP любит ссылаться на самого себя».)

Кстати, пока не забыли, нужно внести небольшие изменения в главную страницу нового сайта. Поскольку вы используете сценарий не только для регистрации при входе в систему, но и для создания самой регистрационной страницы, перед тем как углубиться в PHP, сделайте следующую работу:

```
<html>
 <head>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
 </head>

 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="content">
    <div id="home_banner"></div>
    <div id="signup">
      <a href="create_user.html"><img src="../images/sign_me_up.png" /></a>
      <a href="signin.php"><img src="../images/sign_me_in.png" /></a>
    </div>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Код PHP любит ссылаться на самого себя

До сей поры у вас было довольно строгое разграничение между формами, создаваемыми в файлах HTML, и сценариями, которым они передавали данные, создаваемыми на PHP. Однако у вас уже есть множество сценариев, совершающих какие-нибудь запрограммированные «штучки» — регистрацию пользователя в приложении, получение сведений о всех текущих пользователях или просмотр сведений о конкретном пользователе, а затем выводящих целый пакет HTML-кода.

Тогда почему бы также не вернуть весь ход событий назад в формы? Сценарий может вывести форму и направить действие этой формы на... самого себя. Затем при повторном вызове формы он будет видеть, есть ли у него какие-нибудь отправленные данные. Если есть, он получает отправленные формой данные и может делать все, что ему нужно в запрограммированном режиме. Здесь нет ничего магического, только инструкции `if`, направляющие движение в разные стороны. Внутри такой инструкции `if` вы можете даже вывести совершенно другую страницу, возможно, оповещающую пользователя о том, что его данные были приняты.

А если отправленных данных нет, значит, это обычный начальный запрос формы, и, стало быть, должна быть показана форма. Но здесь также есть некоторые полезные преимущества. Вы можете увидеть возможные ошибки или существующие данные из предыдущей отправки и вставить эти значения непосредственно в вашу форму.

Использование отправленных данных — весьма распространенный прием в PHP, один из тех, к которому вам нужно привыкнуть. Даже притом, что поначалу он несколько не умещается в сознании, вскоре вы поймете, что в приложениях, работающих под управлением PHP, случаи, не подходящие для использования сценария PHP, бывают довольно редко. Формы, страницы ошибок, страницы регистрации для входа в приложение и даже страницы приветствий... Для всего этого вы не преминете воспользоваться возможностью, предоставляемой кодом PHP, и вряд ли вам захочется вернуться к прежним временам.

Теперь только немногие из вас, относящиеся к приверженцам схемы «Модель-представление-контроллер» (Model-View-Controller, MVC), будут стоять на своем. HTML внутри сценария, который отправляет данные самому себе, означает полное разрушение стены (или даже какой-то более существенной преграды) между моделью, представлением и контроллером. Но, как вы уже поняли, мы не собираемся создавать настоящую MVC-схему, хорошо работающую на PHP. Вы можете получить некоторое приближение и не отклоняться от него. Не нужно выполнять четкое разделение, которого можно достичь при работе с такими языками, как Ruby или Java (но если задаться целью, в этих языках также можно все перемешать).

С учетом этого нужно просто согласиться с тем, что PHP часто требует от вас пожертвовать реализацией чистой схемы MVC, положив ее на алтарь получения работоспособного кода. Нужно всегда и при любых условиях осуществлять задуманное.

От HTTP-аутентификации к использованию cookie-файлов

Перед тем как углубиться в написание сценария регистрации (назовем его `signin.php`), нужно заняться решением еще одной насущной проблемы. Как позволить пользователю зарегистрироваться для входа в приложение? Отказываясь от появляющейся формы регистрации, вы берете регистрацию в свои руки.

Получить имя пользователя и пароль и проверить их на соответствие данным, хранящимся в базе, не представляет особого труда. Вы можете и должны сделать это в сценарии `signin.php`. Но большой проблемой является наличие этой информации под рукой. При HTTP-аутентификации браузер знал о всех ваших страницах, принадлежащих одной области, и о факте наличия или отсутствия регистрации пользователя в этой области. Поэтому регистрация и получение доступа к странице сценария `show_users.php` означали, что пользователь не должен был регистрироваться, чтобы попасть в сценарий `delete_user.php`. Регистрация уже состоялась для другой страницы той же самой области.

И тут на первый план выходит использование cookie-файлов.

ПРИМЕЧАНИЕ

И здесь сразу же начинаются дежурные шутки о сдобе, конфетах и миллионе других сладостей (cookie с англ. означает пирожное, пирожок, печенье, в общем, некую сладость). Этот странный термин возвращает нас к некоему понятию, называемому волшебным печеньем (magic cookies). Данный термин использовался специалистами старой школы Unix для небольших фрагментов данных, передававшихся вперед-назад между программами.

В любом случае этот термин прижился, поэтому если вы раньше ничего не знали о cookie-файлах в мире программирования, можете посмеиваться, занимаясь программированием при изучении всего остального материала главы.

Что такое cookie-файлы?

В самой cookie-технологии нет ничего магического. Это всего лишь средство, позволяющее хранить отдельные фрагменты информации. У cookie-файла есть имя, значение, являющееся единым информационным фрагментом, и дата истечения срока годности. Когда этот срок истекает, cookie-файл удаляется, и вы больше не можете получить его значение.

Следовательно, у вас может быть cookie-файл по имени `username` и со значением `my_username` и, возможно, еще один cookie-файл по имени `user_id` и со значением `52`. Затем сценарий может проверить наличие cookie-файла `username` и, если он существует, предположить, что пользователь зарегистрировался в приложении. Точно так же сценарий регистрации может установить cookie-файл `username`.

Иными словами, кроме установки cookie-файла вы в первую очередь получаете тот же самый эффект, который был при стандартной аутентификации. Разумеется, создание cookie-файлов находится под вашим контролем, поэтому вы можете

создавать их в своей собственной форме, удалять из своих сценариев (скажем, при отмене регистрации пользователя) и выдавать сообщения на основе состояния cookie-файлов.

ВНИМАНИЕ

Хотя создание cookie-файлов можно контролировать, ваши пользователи могут запросто изменить, удалить и даже создать собственные cookie-файлы. Поэтому они не являются идеальным средством для той информации, которую вы здесь в них храните: для надежных имен пользователей и тому подобных данных.

Именно поэтому у нас еще есть глава 13. И не нужно бояться, даже если вы измените способ использования cookie-файлов. Все, что будет изучено в данной главе, будет играть важную роль в вашем окончательном решении по аутентификации. Существует множество случаев, когда cookie-файлы могут принести ощутимую пользу, и они станут основным инструментом в вашем арсенале средств программирования.

Создание и извлечение cookie-файлов

Вы почти готовы опять перейти к созданию сценариев и получить от этого удовольствие. (Конечно, не так приятно читать о коде, как его создавать.) Нужно научиться создавать cookie-файлы, а затем находить их и получать их значения. Благодаря PHP работать с cookie-файлами так же просто, как и со сверхглобальными переменными `$_SERVER` и `$_REQUEST`, которыми вы уже пользовались.

Для установки cookie-файла нужно просто вызвать функцию `setcookie` с именем cookie-файла и его значением:

```
setcookie("username", "my_username");
```

После установки cookie-файла получить только что установленное для него значение можно с помощью сверхглобальной переменной `$_COOKIE`:

```
echo "Вы зарегистрировались как " . $_COOKIE['username'] . ".";
```

Вот и все. Никаких сложностей. Разумеется, то здесь, то там встречаются разные шероховатости, и к созданию cookie-файлов нужно будет добавить некоторые нюансы. Но если у вас есть функция `setcookie` и переменная `$_COOKIE`, то вы более-менее готовы к работе с этой технологией.

ПРИМЕЧАНИЕ

Об одном из таких нюансов — о значении срока годности cookie-файла — вы, наверное, уже задумывались. Функции setcookie можно передать третий аргумент, о значении которого для установки срока годности мы скоро узнаем.

Регистрация при входе в приложение с использованием cookie-файлов

Вы уже в курсе, что такое cookie-файлы. Вы знаете порядок развития событий в форме регистрации. Теперь настало время написать код. Создайте сценарий `signin.php` и начните с основной схемы:

```
<?php

require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

// Если пользователь зарегистрировался, будет установлен cookie-файл user_id
if (!isset($_COOKIE['user_id'])) {

  // Проверка отправки формы регистрации с username для входа в приложение
  if (isset($_REQUEST['username'])) {
    // Попытка зарегистрировать пользователя
    $username = mysql_real_escape_string(trim($_REQUEST['username']));
    $password = mysql_real_escape_string(trim($_REQUEST['password']));

    // Поиск пользователя

    // Если пользователь не найден, выдача ошибки
  }

  // Часть if, относящаяся к ситуации «Не вошел
  // как зарегистрированный пользователь».
  // Начало страницы. Мы знаем, что здесь нет сообщения об успехе
  // или об ошибке, поскольку происходит всего лишь регистрация для входа
  // в приложение.
  page_start("Регистрация");
?>

<html>
  <div id="content">
    <h1>Регистрация в клубе</h1>
    <form id="signin_form" action="signin.php" method="POST">
      <fieldset>
        <label for="username">Имя пользователя:</label>
        <input type="text"  name="username" id="username" size="20" />
        <br />
        <label for="password">Пароль:</label>
        <input type="password"  name="password" id="password" size="20" />
      </fieldset>
      <br />
      <fieldset class="center">
        <input type="submit" value="Зарегистрироваться" />
      </fieldset>
    </form>
  </div>
  <div id="footer"></div>
 </body>
</html>

<?php
} else {
```

```
  // Обработка случая, когда зарегистрировавшийся пользователь
  // перенаправляется на другую страницу, скорее всего, на show_user.php
}
?>
```

ПРИМЕЧАНИЕ

Вы заметили, что для регистрации пользователя сценарий database_connection.php потребовался, а сценарий app_config.php нет? Вы можете включить app_config.php, поскольку вероятность того, что он когда-нибудь понадобится, довольно высока. Но вы также должны помнить, что вообще-то сам сценарий database_connection.php требует загрузки сценария app_config.php. Поэтому, если необходим сценарий database_connection.php, то с ним заодно будет требоваться и сценарий app_config.php.

Этот сценарий еще далек от завершения, и у него имеются проблемы, но в нем уже довольно много программного кода. Разберем его по фрагментам.

Зарегистрировался пользователь или нет?

Даже при явном входе пользователя на вашу страницу регистрации вы не должны заставлять его входить на нее только после регистрации. Поэтому первое, что нужно сделать (кроме вызова нескольких инструкций `require_once`), — посмотреть, создан ли cookie-файл `user_id`. Если такого файла нет, значит, пользователь не зарегистрировался и события должны развиваться исходя из этого.

```
<?php

require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

// Если пользователь зарегистрировался, должен быть создан
// cookie-файл user_id
if (!isset($_COOKIE['user_id'])) {
```

Это первая подсказка, свидетельствующая о том, что cookie-файлы не слишком отличаются от того, что вы уже использовали: чтобы убедиться в том, что они уже созданы, вы можете использовать функцию `isset`, передавая ей имя cookie-файла. Проще не придумаешь.

Пытается ли пользователь зарегистрироваться?

Если cookie-файл `user_id` не создан, значит, пользователь не зарегистрировался. Стало быть, теперь вам нужно посмотреть, пытается ли он сделать это. Это будет означать, что у вас есть некая информация о запросе. Пользователь мог уже заполнить HTML-форму (код которой находится в сценарии чуть ниже) и отправить эту форму обратно этому же сценарию.

Но это отличается от попытки доступа к данному сценарию без какой-либо информации. В таком случае пользователь должен просто получить обычную

HTML-форму регистрации. Итак, вы можете посмотреть, есть ли отправленная информация, проверив, присутствуют ли в сверхглобальной переменной $_REQUEST какие-нибудь данные с ключом username, что будет означать ввод данных в соответствующее поле формы регистрации:

```
// Проверка отправки данных формы регистрации по имени пользователя
if (isset($_REQUEST['username'])) {
  // Попытка зарегистрировать пользователя
  $username = mysql_real_escape_string(trim($_REQUEST['username']));
  $password = mysql_real_escape_string(trim($_REQUEST['password']));

  // Поиск пользователя

  // Если пользователь не найден, выдача ошибки
}
```

При наличии данных запроса вы можете получить переданные имя пользователя и пароль и сразу же приступить к поиску пользователя и к решению любых задач.

Но перед этим нужно внести небольшое изменение. До сих пор повсеместно использовалась переменная $_REQUEST. Она вбирала в себя GET-запросы, в которых информация передавалась через URL, и POST-запросы, к которым относились запросы, выдаваемые большинством ваших форм. Но вы уже знаете, что единственный способ получения информации на этой стадии — это ее отправка из вашей собственной формы, которая будет использовать POST-запрос.

Следовательно, вы можете заменить $_REQUEST более конкретной сверхглобальной переменной $_POST, в которой будут содержаться только те данные, которые были отправлены POST-запросом.

ПРИМЕЧАНИЕ

Как вы уже, наверное, догадались, у переменной $_POST имеется аналог и для GET-запросов: переменная $_GET.

Переход на использование по возможности более конкретной переменной $_POST имеет вполне определенный смысл. POST-данные не допускают применения параметров в URL запроса и в общем являются немного более безопасными.

ВНИМАНИЕ

Акцент здесь нужно сделать на слове «немного». POST-данные добыть сложнее, чем GET-данные, но ненамного. Просто не нужно думать, что форма, которая передает данные с помощью POST-запросов, безопасна сама по себе. Это далеко не так.

Внесите в сценарий следующую небольшую правку:

```
// Установка факта отправки формы с именем пользователя для регистрации
if (isset($_POST['username'])) {
  // Попытка зарегистрировать пользователя
  $username = mysql_real_escape_string(trim($_REQUEST['username']));
```

```
  $password = mysql_real_escape_string(trim($_REQUEST['password']));

  // Поиск пользователя

  // Если пользователь не найден, выдача ошибки
}
```

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Какая разница, что использовать: $_REQUEST или $_POST?

Это еще одна тема, вызывающая споры, обвинения и искажения сути вопроса со стороны программистов. Неважно, что вы слышали, но с точки зрения получения информации о запросе между `$_REQUEST`, `$_GET` и `$_POST` нет никакой функциональной разницы. Переменная `$_REQUEST` всегда будет содержать ту информацию, которая имеется как в переменной `$_GET`, так и в переменной `$_POST`. Но если вы знаете, что получен POST-запрос, то вы ничего не приобретете и не потеряете от того, что будете использовать `$_REQUEST`, а не `$_POST`.

На самом деле в переменной `$_REQUEST` собраны не только значения из `$_GET` и `$_POST`. В ней также содержатся значения переменной `$_COOKIE` (по крайней мере по умолчанию). Следовательно, с технической точки зрения в сценарии `signin.php` можно сделать следующее:

```
// Если пользователь зарегистрировался, будет создан cookie-файл user_id
if (!isset($_REQUEST['user_id'])) {
```

Итак, вы можете использовать `$_REQUEST` и полностью отказаться от `$_GET`, `$_REQUEST` и `$_COOKIE`. Но нужно вернуться к мыслям о всех принципах программирования, которых нужно придерживаться: сохранять код понятным и легко читаемым, указывать в нем конкретные, а не только универсальные приемы и думать о том, что будет видеть тот, кто будет работать с вашим кодом после вас. По всем этим причинам можно, конечно, применять и `$_REQUEST`, но полезнее все же по мере востребованности использовать `$_GET`, `$_POST` и `$_COOKIE`.

Что же касается сценария `signin.php`, вы знаете, что будет получен POST-запрос. Исходя из этого, там, где возможно, следует применять переменную `$_POST`. Если известно о получении GET-запроса, нужно использовать переменную `$_GET`. А если вас интересует содержимое cookie-файла, необходимо задействовать переменную `$_COOKIE`. Ваш код будет выглядеть понятнее и конкретнее... и вы будете точно знать, для чего он предназначен.

Отображение страницы

Когда пользователь попадет на эту страницу в результате отправки неверных полномочий или вообще без отправки этих полномочий, он должен увидеть форму. Следовательно, теперь вы готовы отобразить некое HTML-содержимое.

ПРИМЕЧАНИЕ

Если пользователь прошел успешную регистрацию, вашему коду нужно перенаправить его в какое-нибудь другое место. Поэтому блок кода, который проверяет имя пользователя и пароль, в случае успешной регистрации должен в конечном счете направить пользователя в другое место.

```
  // Часть if, относящаяся к ситуации «Не вошел
  // как зарегистрированный пользователь».
  // Начало страницы. Мы знаем, что здесь нет сообщения об успехе или
  // об ошибке, поскольку происходит всего лишь регистрация для входа
  // в приложение.
  page_start("Регистрация");
?>

<html>
  <div id="content">
    <h1>Регистрация в клубе</h1>
    <form id="signin_form" action="signin.php" method="POST">
      <fieldset>
        <label for="username">Имя пользователя:</label>
        <input type="text"  name="username" id="username" size="20" />
        <br />
        <label for="password">Пароль:</label>
        <input type="password"  name="password" id="password" size="20" />
      </fieldset>
      <br />
      <fieldset class="center">
        <input type="submit" value="Зарегистрироваться" />
      </fieldset>
    </form>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

Не пропустите этот открывающий блок комментариев, поскольку он играет важную роль. Данный код, включая HTML, все еще остается частью открытого блока if:

```
// Если пользователь зарегистрировался, должен быть создан
// cookie-файл user_id
if (!isset($_COOKIE['user_id'])) {
```

Иными словами, весь этот HTML будет показан только в том случае, если пользователь не зарегистрировался.

Здесь можно внести еще одно усовершенствование в том же духе, что и использование $_POST вместо $_REQUEST. Посмотрите на эту строку:

```
<form id="signin_form" action="signin.php" method="POST">
```

В этой строке форме предписывается отправка данных тому же сценарию, который генерирует форму. Здесь все правильно. Но что будет, если вы переименуете сценарий signin.php? Возможно, вероятность этого невелика, но все же она вполне реальна. (Совсем недавно вы отказались от вызова сценария admin.php и перешли к более функциональным именам сценариев delete_user.php и show_users.php.)

Следует помнить, что в PHP приветствуется эта парадигма отправки из сценария данных этому же сценарию. Чтобы упростить ситуацию, у вас есть свойство в массиве $_SERVER, сообщающее об имени текущего сценария. Оно находится там не только для сценариев, ссылающихся на самих себя, но в данном случае будет использоваться именно таким сценарием. Обновите signin.php для получения выгоды от $_SERVER['PHP_SELF']:

```
<form id="signin_form"
      action="<?php echo $_SERVER['PHP_SELF']; ?>"
      method="POST">
```

Теперь форма отправляет данные буквально самой себе. Пусть небольшое, но очень полезное изменение, которым вы будете пользоваться еще много раз.

Перенаправление по мере необходимости

Теперь осталось только, по крайней мере в этой псевдокодовой версии, перенаправить пользователя в другое место в случае его успешной регистрации:

```
<?php
} else {
  // Обработка случая, когда зарегистрировавшийся пользователь
  // перенаправляется на другую страницу, скорее всего, на show_user.php
}
?>
```

Итак, теперь вы попадаете в основное течение программы. Но здесь еще масса недостающего кода. Настало время заняться этим вплотную и приступить к переводу кода в работоспособную форму.

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Псевдокод с комментариями и настоящий код

Как бы ни было странно полагать, что сейчас сценарий signin.php существует в виде псевдокода, так оно и есть. Он действительно не является полностью работоспособным сценарием, поскольку в нем повсюду есть дыры, через которые может проехать целый грузовик. Но эти дыры, как правило, обозначены полезным и понятным комментарием. И хотя данные комментарии сами по себе ничего не делают, они напоминают вам о том, что должно быть сделано и где это должно быть сделано.

По правде говоря, зачастую лучше воспользоваться именно таким псевдокодом. При этом не приходится понапрасну тратить время на запись имен несуществующих функций вроде check_the_user_credentials(). Та же самая цель достигается следующими комментариями:

```
// Поиск пользователя

// Если пользователь не найден, выдача ошибки
```

Такие комментарии не менее полезны. Они могут оставаться после написания под каждым из них кода, заполняющего пробелы в функциональности сценария.

Перед тем как приступить к созданию кода, у вас уже должна быть продуктивная идея о ходе выполнения сценария. На данный момент пользователь, не прошедший регистрацию, получит HTML-вывод без всего PHP, запускающегося, когда имя пользователя проходит через POST-запрос. На рис. 12.3 показано, так сказать, исходное представление. Это может показаться немного странным, но один и тот же PHP-сценарий, проверяющий полномочия при регистрации, теперь получает их от ваших пользователей. Это весьма простая HTML-форма, выводимая потому, что отсутствует cookie-файл user_id и в POST-данных нет имени пользователя.

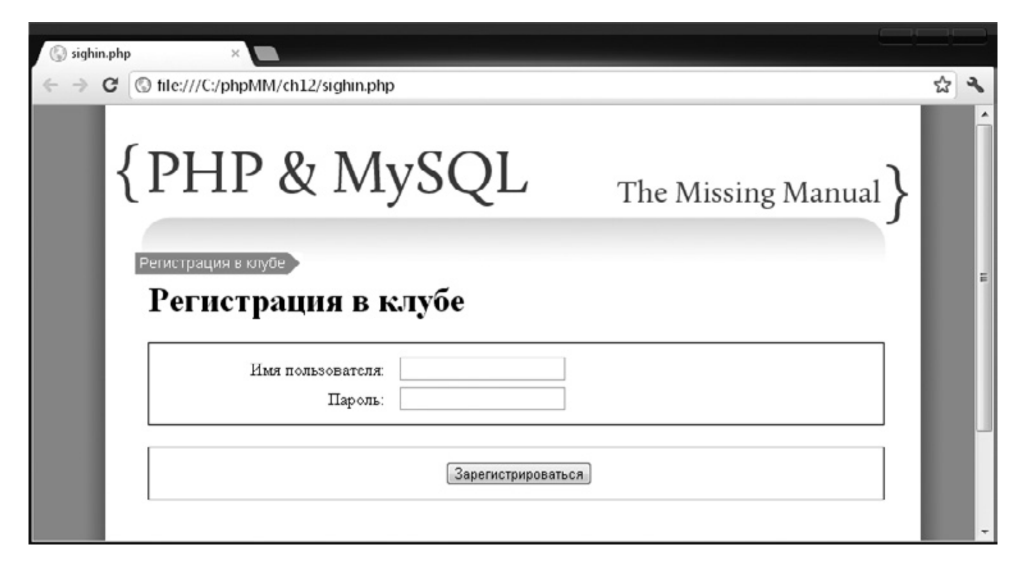


Рис. 12.3. Форма, предлагающая зарегистрироваться

Воспользуйтесь формой и отправьте ее данные с реальным или несуществующим именем пользователя, и вы снова получите точно такую же форму. Не лучший вариант, но все же он может стать отправной точкой. Теперь можно приступать к разработке каждой отдельной функции.

Регистрация пользователя при входе в приложение

Следующий фрагмент кода можно получить путем копирования его из сценария authorize.php с последующей вставкой и правкой. Вот как выглядел этот сценарий:

```
// Поиск предоставленных пользователем полномочий
$query = sprintf("SELECT user_id, username FROM users " .
                 " WHERE username = '%s' AND " .
                 "       password = '%s';",
            mysql_real_escape_string(trim($_SERVER['PHP_AUTH_USER'])),
            mysql_real_escape_string(
              crypt(trim($_SERVER['PHP_AUTH_PW']),
                    $_SERVER['PHP_AUTH_USER'])));

$results = mysql_query($query);

if (mysql_num_rows($results) == 1) {
  $result = mysql_fetch_array($results);
  $current_user_id = $result['user_id'];
  $current_username = $result['username'];
} else {
  header('HTTP/1.1 401 Unauthorized');
  header('WWW-Authenticate: Basic realm="The Social Site"');
  exit("Здесь нужно указать верное имя пользователя и пароль." .
       "Проходите дальше. Здесь вам нечего смотреть.");
}
```

Неплохо, хотя все в нем зависит от HTTP-аутентификации. Теперь этот код можно поместить в сценарий signin.php, изменив блок, выполняющийся в случае успеха для создания cookie-файлов и перенаправления, на какую-нибудь полезную страницу:

```
<?php

require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

// Если пользователь зарегистрировался, должен быть создан
// cookie-файл user_id
if (!isset($_COOKIE['user_id'])) {

  // Установка факта отправки формы с именем пользователя для регистрации
  if (isset($_POST['username'])) {
  // Попытка зарегистрировать пользователя
    $username = mysql_real_escape_string(trim($_REQUEST['username']));
    $password = mysql_real_escape_string(trim($_REQUEST['password']));

    // Поиск пользователя
    $query = sprintf("SELECT user_id, username FROM users " .
```

```
                    " WHERE username = '%s' AND " .
                    "       password = '%s';",
                    $username, crypt($password, $username));

    $results = mysql_query($query);

    if (mysql_num_rows($results) == 1) {
      $result = mysql_fetch_array($results);
      $user_id = $result['user_id'];
      setcookie('user_id', $user_id);
      setcookie('username', $result['username']);
      header("Location: show_user.php");
    } else {
      // Если пользователь не найден, выдача ошибки
    }
  }

  // Часть if, относящаяся к ситуации «Не вошел
  // как зарегистрированный пользователь».
  // Начало страницы. Мы знаем, что здесь нет сообщения об успехе или
  // об ошибке, поскольку происходит всего лишь регистрация для входа
  // в приложение
  page_start("Регистрация");

?>
```

При открытии страницы, создаваемой сценарием signin.php, вы должны получить форму регистрации (посмотрите на рис. 12.2, чтобы убедиться, что вы находитесь на правильной странице с правильным HTML). Зарегистрируйтесь с предоставлением каких-либо подходящих полномочий, и вы попадете в приложение. Для вас будет создан cookie-файл, и вы будете перенаправлены на страницу сценария show_user.php (рис. 12.4).

Вы ничего странного не заметили в последнем перенаправлении? Посмотрите на строку, где перенаправление отправляется браузеру:

```
    if (mysql_num_rows($results) == 1) {
      $result = mysql_fetch_array($results);
      $user_id = $result['user_id'];
      setcookie('user_id', $user_id);
      setcookie('username', $result['username']);
      header("Location: show_user.php");
    } else {
      // Если пользователь не найден, выдача ошибки
    }
```

Если ничего не заметили, посмотрите на прежнюю версию сценария create_user.php. Он создает запись пользователя и перенаправляет этого пользователя на страницу сценария show_user.php. Вот относящаяся к этому строка кода:

```
header("Location: show_user.php?user_id=" . mysql_insert_id());
```



Рис. 12.4. Перенаправление на страницу сценария show_user.php с показом информации о пользователе, который зарегистрировался в приложении

Согласно ей, браузеру в виде GET-параметра внутри URL-запроса отправляется дополнительная информация: `user_id` отображаемого пользователя. Но в сценарии `signin.php` нет параметра `user_id`. И все же на рис. 12.4 показано, что все работает.

Сценарий `show_user.php` ожидает следующую информацию:

```
// Получение ID отображаемого пользователя
$user_id = $_REQUEST['user_id'];
```

И это не единственное место, где сценарию `show_user.php` требуется информация о пользователе. В начале сценария `signin.php` есть следующая открывающаяся инструкция `if`:

```
// Если пользователь зарегистрировался, должен быть создан
// cookie-файл user_id
if (!isset($_COOKIE['user_id'])) {
```

Инструкция `else`, относящаяся к этой инструкции `if`, находится ближе к концу сценария:

```
} else {
  // Обработка случая, когда зарегистрировавшийся пользователь
```

```
  // перенаправляется на другую страницу, скорее всего, на show_user.php
  header("Location: show_user.php");
}
```

Следовательно, здесь еще одно место, из которого пользователь перенаправляется на страницу сценария show_user.php. Здесь нет параметра user_id, но точно так же сценарий show_user.php продолжает успешно определять, какой именно пользователь должен быть показан, и отображает этого пользователя (см. рис. 12.4).

Итак, как же это все функционирует в сценарии signin.php? Ответ кроется в особенностях работы переменной $_REQUEST и той информации, которая в ней содержится (подробнее об этом было сказано во врезке «Часто задаваемый вопрос: какая разница, что использовать: $_REQUEST или $_POST?» выше). В сценарии signin.php вы создаете cookie-файл, который доступен в сверхглобальной переменной $_COOKIE. Но переменная $_REQUEST также содержит информацию, имеющуюся в переменной $_COOKIE, наряду со сведениями, которые имеются в переменных $_POST и $_GET. Поэтому данная строка кода:

```
$user_id = $_REQUEST['user_id'];
```

дает то же самое, что и эта строка:

```
$user_id = $_COOKIE['user_id'];
```

и получает значение, хранящееся в cookie-файле.

ПРИМЕЧАНИЕ

Возникает резонный вопрос: что использовать: $_COOKIE или $_REQUEST? Обычно все зависит от обстоятельств. Здесь, если вы переключаетесь на $_COOKIE, вам нужно обновить сценарий create_user.php. Возможно, будет лучше по-прежнему использовать $_REQUEST, поскольку это придает сценарию show_user.php немного больше гибкости. Эта переменная принимает параметры запроса и содержимое cookie-файлов, и это хорошо. Чуть позже, если вы перейдете к использованию одних только cookie-файлов, можно будет обновить show_user.php, чтобы применять только $_COOKIE, и конкретизировать совершаемые действия.

Пустые страницы и истечение срока действия cookie-файлов

В отдельные моменты опробования кода можно получить весьма странные ответы. Вы набираете в адресной строке signin.php, нажимаете клавишу Enter и получаете пустую страницу (рис. 12.5).

Вы пробуете еще раз, пробуете перезагрузить страницу, пробуете очистить кэш-память. Безрезультатно! И наконец, вы перезапускаете браузер, и все начинает получаться. Но как только вы регистрируетесь через сценарий signin.php, как все повторяется. В чем же дело?

Вообще-то это признак того, что все работает правильно. Вспомните, что в вашем сценарии сначала проверяется условие наличия cookie-файла:

```
// Если пользователь зарегистрировался, должен быть создан
// cookie-файл user_id
if (!isset($_COOKIE['user_id'])) {
```

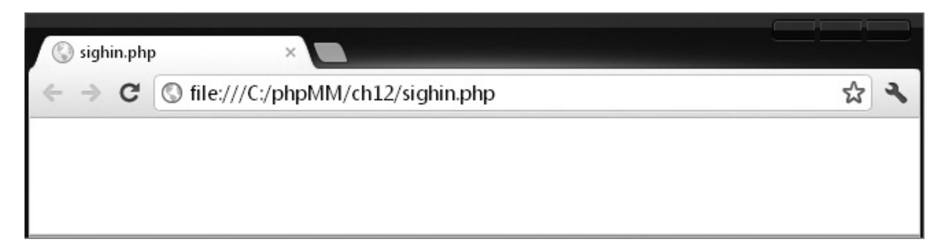


Рис. 12.5. Пустая страница после регистрации

Если этот cookie-файл создан, сценарий сразу переходит к следующему коду, находящемуся в нижней части файла:

```
<?php
} else {
  // Обработка случая, когда зарегистрировавшийся пользователь
  // перенаправляется на другую страницу, скорее всего, на show_user.php
}
?>
```

Но ведь здесь ничего нет! Поэтому вы получаете пустой экран браузера. Вы можете исправить положение дел (отчасти), установив действие по умолчанию для тех пользователей, которые были направлены на страницу сценария `signin.php` и уже зарегистрировались. По сути, это то же самое, что вы делали раньше для первичного внесения данных: перенаправляли пользователя на страницу сценария `show_user.php`:

```
} else {
  // Обработка случая, когда зарегистрировавшийся пользователь
  // перенаправляется на другую страницу, скорее всего, на show_user.php
  header("Location: show_user.php");
}
```

Теперь уже нет пустого экрана. Сценарий `show_user.php` заберет cookie-файл `user_id` и покажет только что зарегистрировавшегося пользователя. Неплохо?

Отчасти. Вы по-прежнему находитесь в бесконечном цикле. Только теперь вы просматриваете в цикле страницу сценария `show_user.php`, а не безликую пустую страницу. Чтобы остановить это сумасшествие, нужно полностью закрыть браузер.

Но это вполне нормальное поведение. Так же, как и при регистрации с помощью HTTP-аутентификации, регистрация здесь и создание cookie-файла по умолчанию определяют существование этого cookie-файла до тех пор, пока браузер не будет закрыт.

ПРИМЕЧАНИЕ

Вызов функции setcookie со значением третьего аргумента, установленного по умолчанию в «0», означает, что срок действия cookie-файла истечет в конце рабочей сессии пользователя, то есть когда пользователь зароет свой браузер.

Поэтому если нужно выбраться из этого цикла, следует просто закрыть браузер. Убедитесь в том, что вы закрыли программу, а не просто вкладку или окно. Это заставит cookie-файл с установленным по умолчанию сроком действия закончить работу.

Если же вам нужно создать cookie-файл с более продолжительным (или более коротким сроком действия), то просто передайте функции `setcookie` третий аргумент. Он должен быть выражен в секундах, прошедших с момента отсчета компьютерной эры в системах Unix и Linux, с 0:00 по Гринвичу 1 января 1970 года. Поэтому обычно функции передается текущее время, для удобства выраженное в секундах, прошедших с начала эры, с некоторым добавлением. Следовательно, `time() + 10` будет означать 10 секунд в будущем, вычисляемые с начала эры.

Рассмотрим несколько примеров функции `setcookie` с переданным ей временем истечения срока действия cookie-файла:

```
// Срок истекает через час (60 seconds * 60 minutes = 3600)
setcookie('user_id', $user_id, time() + 3600);

// Этот код просто удаляет cookie-файл, поскольку в нем указано уже
// прошедшее время истечение срока
setcookie('user_id', $user_id, time() - 3600);

// Установка по умолчанию: срок истекает с закрытием браузера
setcookie('user_id', $user_id, 0);
```

Время можно предоставить также через функцию `mktime`, которой передаются час, дата, секунда, месяц, день и год, а она возвращает количество секунд, прошедших с начала компьютерной эры. Следующий код:

```
setcookie('user_id', $user_id, mktime(0, 0, 0, 2, 1, 2021);
```

установит срок истечения действия cookie-файла на 1 февраля 2021 года, полночь по Гринвичу. Вы, наверное, скажете, что это весьма отдаленный срок. Вообще-то вполне резонно воспользоваться сроком, устанавливаемым по умолчанию. Большинство пользователей вполне соглашаются с необходимость повторной регистрации после закрытия своих браузеров. Вряд ли кому понравится, если их регистрация будет действовать вечно.

ПРИМЕЧАНИЕ

Исключением здесь можно назвать такие сайты, как Facebook и Twitter, которые не содержат большого объема особо ценной пользовательской информации. А вот большинство финансовых сайтов не дожидается даже закрытия браузера, завершая сессию примерно каждые 10 минут.

Итак, закройте браузер, завершив тем самым работу cookie-файлов, и еще раз откройте файл `signin.php` для внесения в него некоторых улучшений.

Ошибки не всегда должны прерывать работу приложения

Теперь вы получили потенциальную ошибку, с которой нужно справиться. Этот тот самый блок else, который выполняется, когда имя пользователя и пароль не соответствуют имеющимся в базе данных:

```
if (mysql_num_rows($results) == 1) {
  // Создание cookie-файла и перенаправление на show_user.php
} else {
  // Если пользователь не найден, выдача ошибки
}
```

До сих пор обработкой ошибок занимался сценарий handle_error. Но нам это не подходит, поскольку вы не хотите сорвать процесс регистрации, выбрасывая пользователя на страницу ошибки. Он должен будет вернуться на страницу регистрации, попробовать зарегистрироваться еще раз и потенциально снова окажется на странице ошибки. Для вас это совершенно неприемлемо.

Вам нужно средство, позволяющее показывать любую ошибку, не прерывая общий ход работы приложения. Когда дела совсем плохи, то вполне разумно воспользоваться сценарием handle_error. Серьезная ошибка является вполне веской причиной для прерывания работу приложения. Здесь нужен способ показа ошибки, не прерывающий работу приложения и отличающийся от работы сценария handle_error.

Но у вас уже есть другой способ показа ошибки: функция page_start в сценарии view.php. В данный момент вы вызываете эту функцию в signin.php, но без каких-либо аргументов, кроме заголовка страницы:

```
page_start("Регистрация");
```

Но посмотрим еще раз на сценарий view.php, здесь эта функция получает полный комплект аргументов:

```
function page_start($title, $javascript = NULL,
                    $success_message = NULL, $error_message = NULL) {
```

Обычно требуемые аргументы передаются в виде значений для переменных $success_message и $error_message. Но обязательным условием это не является. То есть вы можете создать новую переменную по имени $error_message, присвоить ей где-нибудь в сценарии текстовое значение, а затем передать ее функции page_start, как только начнется HTML-вывод.

Вот что нужно сделать:

```
<?php

require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

$error_message = "";

// Если пользователь зарегистрировался, должен быть создан
```

```
// cookie-файл user_id
if (!isset($_COOKIE['user_id'])) {
  // Установка факта отправки формы с именем пользователя для регистрации
  if (isset($_POST['username'])) {
    // Попытка зарегистрировать пользователя

    // Поиск пользователя

    if (mysql_num_rows($results) == 1) {
      $result = mysql_fetch_array($results);
      $user_id = $result['user_id'];
      setcookie('user_id', $user_id);
      setcookie('username', $result['username']);
      header("Location: show_user.php");
    } else {
      // Если пользователь не найден, выдача ошибки
      $error_message = "Вы дали неверную комбинацию имя пользователя-пароль.";
    }
  }

  // Часть if, относящаяся к ситуации «Не вошел
  // как зарегистрированный пользователь».
  // Начало страницы и передача любого, установленного ранее
  // сообщения об ошибке.
  page_start("Регистрация", NULL, NULL, $error_message);
?>

<!-- Весь остальной HTML-вывод -->

<?php
} else {
  // Обработка случая, когда зарегистрировавшийся пользователь
  // перенаправляется на другую страницу, скорее всего, на show_user.php
  header("Location: show_user.php");
}
?>
```

ВНИМАНИЕ

Следует помнить, что это решение на основе использования cookie-файлов — шаг навстречу окончательному решению, но не само это решение. В следующей главе будет добавлена поддержка сессий и такая информация, как имя пользователя и ID пользователя, будет перемещена из пользовательских cookie-файлов на сервер.

Но как бы то ни было, продолжайте читать эту главу! Изученная здесь работа с cookie-файлами вам еще пригодится, а в следующей главе к этим навыкам добавятся навыки работы с сессиями. Горе тем программистам, которые используют для аутентификации исключительно cookie-файлы.

Теперь зайдите на страницу сценария `signin.php` (или `index.html`) и щелкните на кнопке регистрации. Представшая перед вами картина, показанная на рис. 12.6, свидетельствует о том, что где-то все же возникла проблема.

Рис. 12.6. В области сообщения об ошибке отсутствует само сообщение

При написании приложений проблемы возникают практически всегда. Вы создали функцию, показывающую ошибку давным-давно, еще в коде сценария `view.php`, а затем намного позже использовали ее другим образом. Именно здесь и произошла ошибка.

В данном случае проблема состоит в том, что вы вызвали функцию `page_start` с переменной `$error_message`, но в некоторых случаях эта переменная содержит пустую строку, "", и поэтому ничего не будет показано. Проверьте код сценария `view.php` и найдите функцию `display_message`:

```
function display_messages($success_msg = NULL, $error_msg = NULL) {
  echo "<div id='messages'>\n";
  if (!is_null($success_msg)) {
    display_message($success_msg, SUCCESS_MESSAGE);
  }
  if (!is_null($error_msg)) {
    display_message($error_msg, ERROR_MESSAGE);
  }
  echo "</div>\n\n";
}
```

В данном случае переменная $error_message не является не содержащей вообще никакого значения. Она содержит пустую строку. Поэтому если ее передать в таком виде, будет показано пустое сообщение об ошибке. Нам такой вариант не подходит.

Исправить положение довольно легко, нужно просто посмотреть, есть ли у $error_message значение, отличное от null, и имеет ли оно длину больше нуля. И попутно нужно внести такое же усовершенствование в обработку сообщений об успешном завершении действия:

```
function display_messages($success_msg = NULL, $error_msg = NULL) {
  echo "<div id='messages'>\n";
  if (!is_null($success_msg) && (strlen($error_msg) > 0)) {
    display_message($success_msg, SUCCESS_MESSAGE);
  }
  if (!is_null($error_msg) && (strlen($error_msg) > 0)) {
    display_message($error_msg, ERROR_MESSAGE);
  }
  echo "</div>\n\n";
}
```

Теперь можно будет увидеть вполне нормальную форму регистрации (рис. 12.7).

Попробуйте ввести неверное имя пользователя или пароль, и вы увидите красивое и понятное сообщение об ошибке, которое не удаляет вас из процесса регистрации. Это сообщение, позволяющее пользователю немедленно ввести заново свои данные, показано на рис. 12.8.

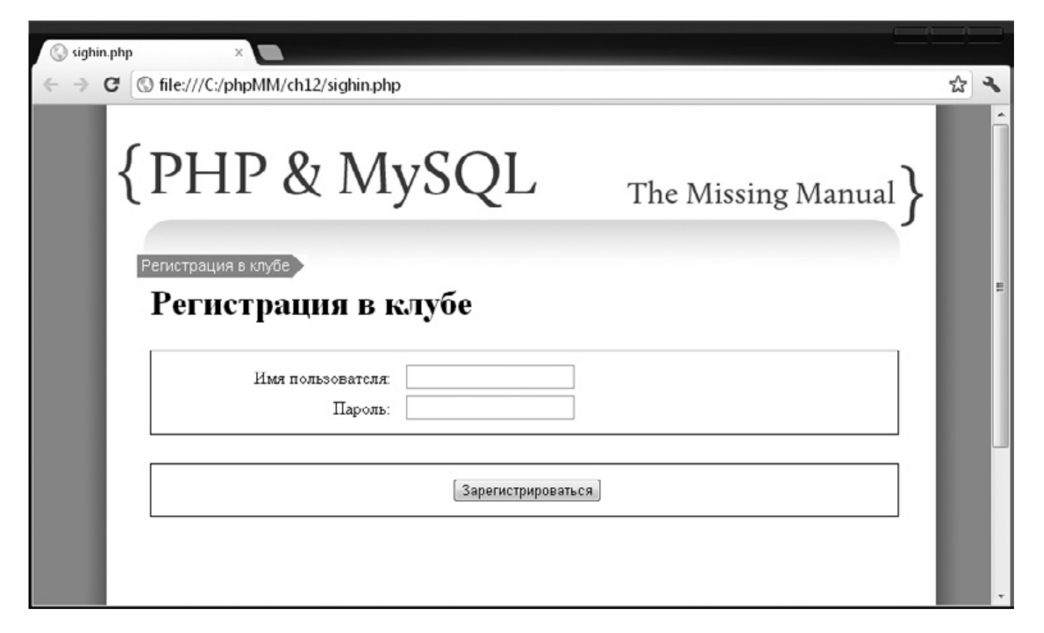


Рис. 12.7. Нормальная форма регистрации



Рис. 12.8. Компактное сообщение об ошибке, не прерывающее рабочий процесс

Настройка на повторные попытки

На данный момент страница регистрации приобрела функционально завершенный вид. Но осталась еще одна возможность, которую нужно предоставить вашим пользователям: перезагрузка их имени пользователя при неудачной регистрации. Некоторые сайты выполняют такую перезагрузку, а некоторые нет. Это вопрос вкуса, но, как и в большинстве случаев, даже если вы решите не предоставлять это свойство, вы должны знать, как его можно реализовать.

Демонстрация имени пользователя предполагает хотя бы однократную предварительную отправку данных формы, после которой управление передается в следующее место сценария `signin.php`:

```
if (isset($_POST['username'])) {
    // Попытка зарегистрировать пользователя
    $username = mysql_real_escape_string(trim($_REQUEST['username']));
    $password = mysql_real_escape_string(trim($_REQUEST['password']));

    // и т. д...
}
```

Имя пользователя было отправлено, но регистрация не состоялась. Следовательно, вы получили готовую к показу переменную $username.

Теперь перейдите к HTML. Вы можете создать значение формы поля с атрибутом value и получить значение этого атрибута в переменной $username. Объедините все это, и получится примерно следующий код:

```
<label for="username">Имя пользователя:</label>
<input type="text"  name="username" id="username" size="20"
      value="<?php if (isset($username)) echo $username; ?>" />
```

Вот, собственно, и все. Введите имя пользователя, отправьте форму регистрации, и вы увидите страницу регистрации с сообщением об ошибке и с введенным ранее именем пользователя (рис. 12.9).

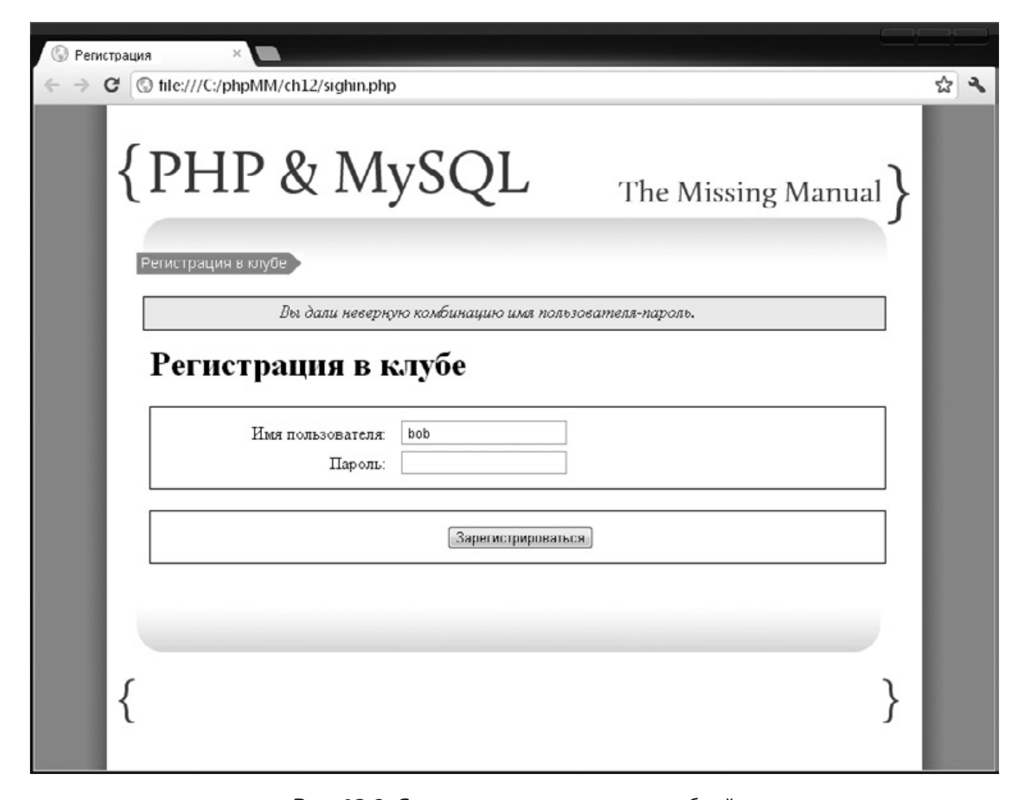


Рис. 12.9. Страница регистрации с ошибкой

Добавление контекстно-зависимых меню

Теперь настало время уделить больше внимания меню и навигации. Можно говорить о множестве существующих конструкций пользовательского интерфейса и удобстве их применения, обсуждать принципы дизайна и часто оспариваемые за и против горизонтальных меню по сравнению с вертикальными. Но непосредственно к PHP

это все не имеет никакого отношения. Вас как PHP-программиста должно интересовать создание ссылок и возможностей, изменяющихся на основе регистрации пользователя.

Вы уже знаете, как увидеть, зарегистрировался пользователь или нет. Вы можете просто проверить наличие cookie-файла user_id:

```
if (isset($_COOKIE['user_id'])) {
  // Вывод возможностей для зарегистрировавшихся пользователей
} else {
  // Вывод возможностей для незарегистрировавшихся пользователей
}
```

Этим все сказано.

Установка меню

Вернемся к сценарию view.php, в котором находится весь код, управляющий заголовками на вашей странице. В подобных ситуациях помещение части основного кода представления в сценарии, на которые могут ссылаться все остальные ваши страницы, имеет огромное значение. Сейчас первыми фрагментами отображаемой страницы управляет функция display_title.

Найдите функцию, к которой можно добавить очень простое условие if: если существует cookie-файл user_id, необходимо показать заметную ссылку на страницу сценария show_user.php и ссылку на страницу сценария signout.php (о которой речь еще впереди). Если пользователь не зарегистрировался, ему нужно показать ссылку на страницу регистрации. И, разумеется, независимо от обстоятельств, необходимо отобразить ссылку на главную страницу:

```
function display_title($title, $success_message = NULL, $error_message = NULL)
{
echo <<<EOD
 <body>
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">$title</div>
  <div id="menu">
    <ul>
      <li><a href="index.html">Главная страница</a></li>
EOD;
  if (isset($_COOKIE['user_id'])) {
    echo "<li><a href='show_user.php'>Мой профиль</a>";
    echo "<li><a href='signout.php'>Отмена регистрации</a></li>";
  } else {
    echo "<li><a href='signin.php'>Регистрация</a></li>";
  }
echo <<<EOD
    </ul>
  </div>
EOD;
  display_messages($success_message, $error_message);
}
```

У этого кода есть две сильные стороны. Во-первых, теперь данное меню доступно всем сценариям через вызов сценария view.php. Следовательно, вам не нужно прокладывать маршрут по всем своим файлам и вставлять в них новый HTML и инструкции if для получения меню, распространяющие свое действие на весь сайт. И во-вторых, поскольку вы поместили этот код в функцию display_title, любые ваши сценарии, которые уже вызывают функцию display_title, автоматически получают код этого меню. Здесь вообще ничего не нужно менять.

И в этот раз сценарий упрощен за счет того, что в переменной $_REQUEST будет возвращено все то же самое, что было возвращено в переменной $_COOKIE:

```
if (isset($_COOKIE['user_id'])) {
  echo "<li><a href='show_user.php'>Мой профиль</a>";
  echo "<li><a href='signout.php'>Отмена регистрации</a></li>";
} else {
  echo "<li><a href='signin.php'>Регистрация</a></li>";
}
```

Нам не нужно заботиться о передаче ID пользователя сценарию show_user.php, потому что создан cookie-файл. И вы уже убедились в том, что сценарий show_user.php может благополучно получить нужное значение из переменной $_REQUEST['user_id'], точно так же, как если бы вы передали ему ID пользователя явным образом через параметр запроса.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Неужели кто-то до сих пор все еще отменяет свою регистрацию в приложении?

Действительно, пока люди не окажутся на каком-нибудь сайте, связанном с финансами — сайте их банка или на биржевом сайте, — регистрация и ее отмена их по большому счету не волнует. Большинство интернет-пользователей не слишком озабочены безопасностью и ждут, что веб-сайт их вспомнит, когда они на него вернутся. Отмена регистрации препятствует этому, но зачем это делается?

Есть весьма веские причины для того, чтобы добавить возможности отмены регистрации в любое приложение. Во-первых, если пользователи получают доступ к вашему приложению на публичном компьютере или на совместно используемом ноутбуке, вам нужно дать им гарантию, что они могут защитить свои полномочия путем отмены регистрации перед тем, как позволить воспользоваться компьютером кому-то другому. Во-вторых, то, что большинство пользователей не слишком заботятся о своей безопасности, еще не свидетельствует о том, что это относится абсолютно ко всем пользователям. Предоставьте кому-нибудь возможность отменить регистрацию. А если они ею не воспользуются, ничего страшного не случится. Зато те, кому это будет необходимо, обрадуются, что ваше приложение поддерживает столь нужную им функцию.

Вы знаете, как создавать cookie-файлы. Будет неплохо узнать и о том, как их можно удалять. Следовательно, добавление ссылки на отмену регистрации заставит вас разобраться и с этой задачей.

Чтобы проверить это в работе, нужно открыть страницы разных сценариев, отображающих HTML: show_user.php, show_users.php и signin.php. Каждый из них должен вызывать функцию page_start, вместо того чтобы показывать HTML в явном виде. В противном случае у вас не будет кода меню, только что добавленного в функцию display_title, которая вызывается из функции page_start в сценарии view.php. Вот как, к примеру, должен выглядеть сценарий show_user.php:

```
<?php

require '../scripts/database_connection.php';
require '../scripts/view.php';

// Большой фрагмент кода PHP для загрузки ID пользователя из параметра
// запроса или из cookie-файла, для поиска этого пользователя
// и для установки некоторых значений

page_start("Профиль");
?>

  <div id="content">
    <div class="user_profile">
      <h1><?php echo "{$first_name} {$last_name}"; ?></h1>
      <p><img src="<?php echo $user_image; ?>" class="user_pic" />
        <?php echo $bio; ?></p>
      <p class="contact_info">
        Поддерживайте связь с <?php echo $first_name; ?>:
      </p>
      <ul>
        <li>... по электронной почте
          <a href="<?php echo $email; ?>"><?php echo $email; ?></a></li>
        <li>... путем
          <a href="<?php echo $facebook_url; ?>">посещения его страницы
             на Facebook</a></li>
        <li>... путем <a href="<?php echo $twitter_url; ?>">отслеживания его
             сообщений в Twitter</a></li>
      </ul>
    </div>
  </div>
  <div id="footer"></div>
 </body>
</html>
```

Теперь зарегистрируйтесь и перейдите на страницу сценария show_user.php.

ПРИМЕЧАНИЕ

Вообще-то вы должны быть автоматически перенаправлены на show_user.php сразу же после успешной регистрации. Это еще лучше!

Вы должны увидеть что-либо подобное изображенному на рис. 12.10. Теперь благодаря функциям start_page, display_title, сценарию view.php и cookie-файлам,

созданным в сценарии `signin.php`, справа на этой странице имеется вполне привлекательное простое меню. Теперь создание подобного меню, когда у вас есть основной механизм его отображения для аутентифицированных пользователей и сокрытия от других пользователей, особого труда не представляет. Вы можете добавить все ссылки или подчиненные ссылки, необходимые вашему приложению, и если они входят в блок `if` в функции `display_title`, требующей наличия cookie-файла, то все должно получиться.



Рис. 12.10. Страница биографии с отдельным меню

От HTML к сценариям

Вы могли заметить, что даже при внесении поправок в `show_user.php`, `show_users.php` и `signin.php`, в вашем приложении еще остались простые страницы. Это `index.html`, начальная страница, а также `create_user.html`. Но данные страницы, конечно же, не пользуются преимуществами функции `start_page` и сценарием `view.php`, поскольку в них содержится код HTML, а не код PHP. Возможно, для `index.html` в этом есть смысл. Из нее вам нужно направить пользователя только в два места: на страницу регистрации или на страницу первичной регистрации, доступ к которым открыт через соответствующие пункты меню.

Но в случае с `create_user.html` совсем другое дело. Представьте, что кто-нибудь щелкнул на пункте меню, чтобы попасть в эту форму, и желает вернуться на главную

страницу. Или, скорее, всего, он может захотеть зарегистрироваться, а не пройти начальную регистрацию. Такая возможность становится еще насущнее по мере добавления к меню дополнительных пунктов, например пункта получения информации о странице.

Любой файл HTML может быть превращен в PHP

Превратить `create_user.html` в `create_user.php` совсем нетрудно. Но постойте, ведь файл `create_user.php` уже существует. Поэтому сначала нужно переименовать `create_user.html` в `signup.php`. Это хорошо согласуется с формулировкой, используемой в `index.html`, и данная форма действительно предназначена для начальной регистрации (sign up) пользователей.

```
[~/www/phpMM/ch12]# cp create_user.html create_user.html.orig
[~/www/phpMM/ch12]# mv create_user.html signup.php
```

ПРИМЕЧАНИЕ

Стоит создать резервные копии исходных файлов, гарантировав тем самым возможность отмены любых внесенных изменений. Это может быть реализовано с помощью полноценной стратегии резервного копирования, распространяющейся на весь сайт, или путем простого дублирования файлов с понятными именами, указывающими на то, что файл имеет отношение к резервной копии.

Затем можно просто удалить открывающий тег HTML и заменить его управляемым с помощью PHP вызовом функции `page_start`. Вам придется пройти через всю внутреннюю JavaScript-проверку, но теперь это не составит особого труда, поскольку вы можете воспользоваться структурой heredoc.

```
<?php

require_once "../scripts/view.php";

$inline_javascript = <<<EOD
    $(document).ready(function() {
      $("#signup_form").validate({
        rules: {
          password: {
            minlength: 6
          },
          confirm_password: {
            minlength: 6,
            equalTo: "#password"
          }
        },
        messages: {
          password: {
            minlength: "Пароль должен иметь не менее 6 символов"
          },
          confirm_password: {
```

```
            minlength: "Пароль должен иметь не менее 6 символов",
            equalTo: "Ваши пароли не совпадают."
          }
        }
      });
    });
EOD;
page_start("Регистрация пользователя", $inline_javascript);
?>

  <div id="content">
    <h1>Вступайте в наш виртуальный клуб</h1>
    <p>Пожалуйста, введите ниже свои данные для связи в Интернете:</p>
    <form id="signup_form" action="create_user.php"
          method="POST" enctype="multipart/form-data">
      <!-- Содержимое формы -->
    </form>
  </div>

  <div id="footer"></div>
 </body>
</html>
```

Теперь настало время обновить файл view.php, чтобы включить в него вызов библиотеки jQuery, сценарии проверки и таблицы CSS, необходимые сценарию signin.php. Неплохо будет также открыть доступ к этим возможностям и для всех остальных страниц сайта:

```
function display_head($page_title = "", $embedded_javascript = NULL) {
echo <<<EOD
<html>
 <head>
  <title>{$page_title}</title>
  <link href="../css/phpMM.css" rel="stylesheet" type="text/css" />
  <link href="../css/jquery.validate.password.css" rel="stylesheet"
        type="text/css" />
  <script type="text/javascript" src="../js/jquery.js"></script>
  <script type="text/javascript" src="../js/jquery.validate.js"></script>
  <script type="text/javascript"
          src="../js/jquery.validate.password.js"></script>
EOD;
  if (!is_null($embedded_javascript)) {
    echo "<script type='text/javascript'>" .
         $embedded_javascript .
         "</script>";
  }
  echo " </head>";
}
```

Обновите ссылку в файле index.html, чтобы она указывала на файл signup.php, а не на файл create_user.html:

```
<div id="content">
  <div id="home_banner"></div>
  <div id="signup">
    <a href="signup.php"><img src="../images/sign_me_up.png" /></a>
    <a href="signin.php"><img src="../images/sign_me_in.png" /></a>
  </div>
</div>
```

Теперь можете проверить новую страницу в работе, на ней должно появиться новое меню. Результаты показаны на рис. 12.11. Эта версия меню относится к «незарегистрировавшемуся» пользователю. Таким образом, теперь вы проверили обе версии, великолепно.

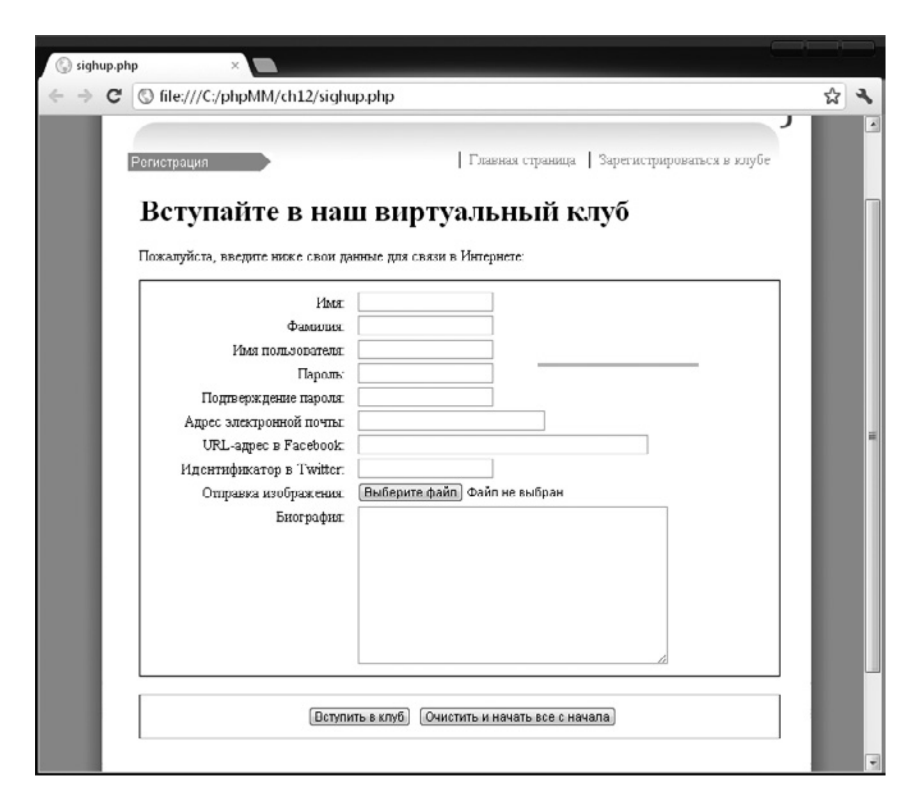


Рис. 12.11. Меню на странице первоначальной регистрации

Задача: создать из сценария создания пользователя ссылку на него самого

Итак, к этому моменту обнаружилось, что вам не нужны два сценария, чтобы проводить начальную регистрацию пользователя. Можно создать один сценарий, отправляющий данные себе самому. Это позволит не только осуществлять на сторо-

не клиента уже имеющуюся проверку приемлемости данных с помощью jQuery и JavaScript, но и проверять имена пользователей и адреса электронной почты на предмет их присутствия в базе данных и возвращать ошибки при наличии дубликатов.

Но на данной стадии вам не нужно просматривать еще один длинный листинг кода. Вы можете создать код самостоятельно. Отложите книгу в сторону и приступайте к объединению файлов signin.php и create_user.php.

Отмена регистрации пользователей

Теперь у вас есть работоспособная система регистрации, но не нужно забывать и об ее отмене. На какой бы срок ни устанавливалось действие cookie-файла (короткий, всего на несколько минут, или длинный), нужно всегда давать пользователю возможность управляет его собственной аутентификацией. Он должен иметь возможность зарегистрироваться или отменить свою регистрацию когда захочет.

Регистрация влечет за собой создание cookie-файла, имеющего имя, значение и необязательный срок истечения действия:

```
setcookie('user_id', $user_id);             // Срок действия по умолчанию
setcookie('username', $result['username']); // Отмена регистрации при
                                            // закрытии браузера
```

Отмена регистрации очень похожа на обратный процесс. Нужно просто указать для cookie-файла пустое значение и установить срок действия в прошедшем времени:

```
// Установка истечения срока действия cookie-файла user_id на год ранее
setcookie('user_id', '', time()-(60*60*24*365));
```

В данном случае cookie-файлу user_id присвоено значение пустой строки, а истечение срока действия назначено на год ранее текущего момента.

ПРИМЕЧАНИЕ

Устанавливайте время истечения срока в прошлом подальше. Тогда, если часы на вашем сервере на пару минут или даже на пару дней отстают, это не повлияет на работу вашего кода. Проблемы возникнут только в том случае, если системное время будет отставать более чем на год.

Превращение этого замысла в сценарий не представляет никакого труда. Просто назначьте сроком действия двух cookie-файлов, используемых вашим приложением — user_id и username, — какой-нибудь момент в далеком прошлом и перенаправьте пользователя на главную страницу или на страницу регистрации:

```
<?php

setcookie('user_id', '', time()-(365*24*60*60));
setcookie('username', '', time()-(365*24*60*60));

header('Location: signin.php');
?>
```

Чтобы испытать это в работе, зайдите в свое приложение (после закрытия браузера и удаления всех cookie-файлов) и зарегистрируйтесь под именем известного пользователя. Вам будет предоставлена возможность посетить страницы сценариев `show_user.php`, `show_users.php` и удаления пользователей. Все работает должным образом.

ПРИМЕЧАНИЕ

Это своего рода рабочий код. Любой старый пользователь может просматривать всех пользователей и удалять пользователей, но вскоре вы исправите ситуацию.

Теперь щелкните в меню на пункте Отмена регистрации. Вы должны быть перенаправлены на страницу регистрации. Вы также можете посетить страницы, требующие ID пользователя, и не увидите своего пользовательского профиля. Это, конечно, хорошо, но то, что получается в результате этого, нам не подходит. Посмотрите на рис. 12.12.

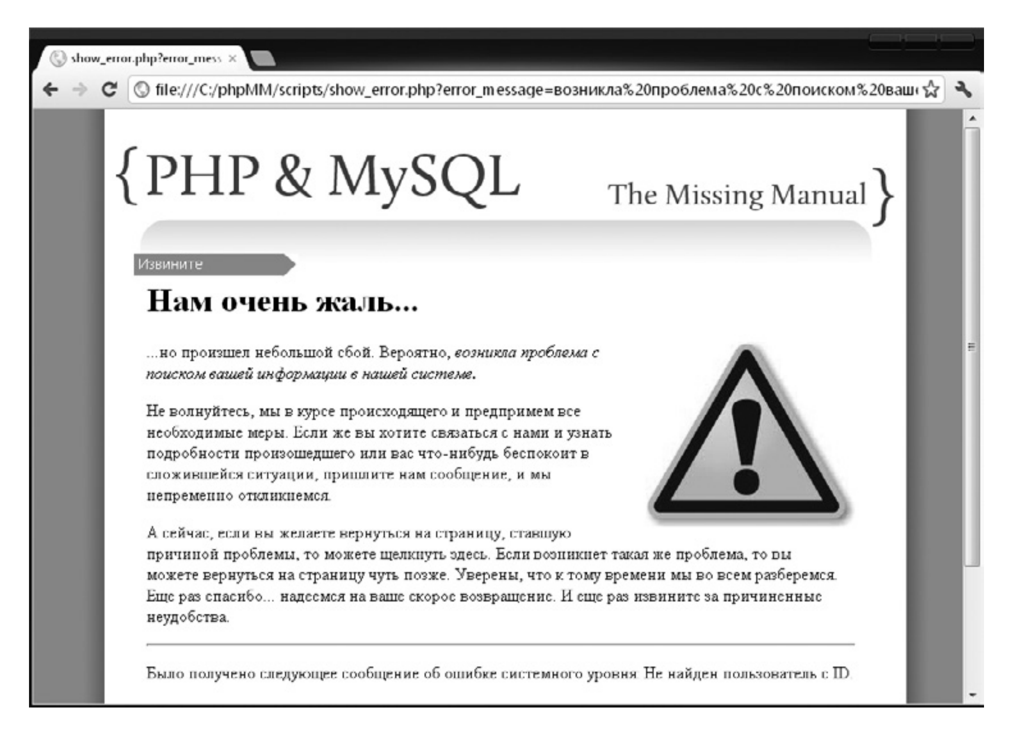


Рис. 12.12. Вместо этого окна пользователю должна быть показана страница регистрации, чтобы пользователь мог зарегистрироваться и получить доступ к системе

Итак, похоже, что отмена регистрации работает, но обнаружился неприятный пробел в приложении: доступность тех страниц, которые вообще-то не должны быть доступны. Они просто выводят сообщение об ошибке, что ничуть не лучше предоставления возможности неавторизованным пользователям просматривать эти страницы. Неважно как, но эту проблему нужно решить.

Требование создания cookie-файла

К счастью, эту проблему довольно легко устранить. Ранее сценарий show_user.php и другие сценарии ограниченного использования требовали для своей работы загрузки сценария authorize.php. Он проводил все необходимые действия с базой данных, чтобы понять, может ли пользователь зарегистрироваться, используя стандартную HTTP-аутентификацию. При этом вокруг сценариев он выстраивал надежную стену безопасности.

После удаления authorize.php появилась возможность управлять регистрацией с помощью сценария signin.php. Однако в процессе этой работы существовавшая вокруг других ваших сценариев стена была разрушена. Вам нужна стена, но вам нужно также оставить управление аутентификацией за сценарием signin.php. Добиться этого нетрудно.

Сначала можно радикально упростить сценарий authorize.php, сократив его функции до проверки наличия подходящего cookie-файла:

```
<?php
if ((!isset($_COOKIE['user_id'])) ||
    (!strlen($_COOKIE['user_id']) > 0)) {
}
?>
```

Если cookie-файла нет или если в нем содержится пустое значение, просто перенаправьте пользователя на страницу регистрации с сообщением, объясняющим происходящее:

```
<?php
if ((!isset($_COOKIE['user_id'])) ||
    (!strlen($_COOKIE['user_id']) > 0)) {
  header('Location: signin.php?' .
         'error_message=You must login to see this page.');
          // Для просмотра этой страницы нужно зарегистрироваться
  exit;
}
?>
```

ВНИМАНИЕ

Инструкция exit играет здесь весьма важную роль. Поскольку этот код будет запускаться, а затем возвращать управление вызывающему сценарию — show_user.php, или delete_user.php, или какому-нибудь другому, — вам нужно гарантировать, что эти сценарии не будут продолжать попытки своего запуска. Нужно отправить перенаправляющие заголовки и устраниться от какого-либо дальнейшего действия.

Отлично, теперь вы можете опять добавить инструкцию require_once в сценарии show_user.php, show_users.php и delete_user.php.

Испытайте проделанную работу на практике. Убедитесь в том, что вы отменили регистрацию (теперь это можно сделать, вызвав сценарий signout.php через

имеющуюся в меню ссылку). Затем попробуйте получить доступ к странице сценария show_user.php. Вы заметите прогресс, хотя здесь не все еще в полном порядке. Но картинка, показанная на рис. 12.13, является неплохой отправной точкой. Теперь попытки доступа к защищенным страницам заканчиваются отправкой на страницу регистрации. Отлично... но где же поясняющее сообщение? Заметьте, в URL запроса оно есть, а на странице не показано.



Рис. 12.13. Попытки доступа к защищенным страницам заканчиваются отправкой на страницу регистрации

Что происходит? Просто в сценарии signin.php нет кода, работающего с потенциальными сообщениями, которые находятся в URL запроса. Но у вас уже был механизм, занимающийся данной проблемой.

Откройте сценарий signin.php и проверьте начальный раздел:

```
require_once '../scripts/view.php';

$error_message = "";

// Если пользователь зарегистрировался, должен быть создан
// cookie-файл user_id
if (!isset($_COOKIE['user_id'])) {
```

Все в порядке. Но у вас уже был код для отображения в качестве сообщения об ошибке значения переменной $error_message:

```
  // Часть if, относящаяся к ситуации «Не вошел
  // как зарегистрированный пользователь».
  // Начало страницы и передача любого, установленного ранее сообщения
  // об ошибке.
  page_start("Регистрация", NULL, NULL, $error_message);
```

Теперь где-то в начале нужно посмотреть, был ли у запроса какой-нибудь параметр:

```
<?php

require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

$error_message = $_REQUEST['error_message'];

// Если пользователь зарегистрировался, должен быть создан
// cookie-файл user_id
if (!isset($_COOKIE['user_id'])) {
```

Проще некуда. Теперь можно попытаться проделать все еще раз. Зайдите на страницу сценария show_user.php без созданного cookie-файла, и вы увидите что-то похожее на изображение, показанное на рис. 12.14.

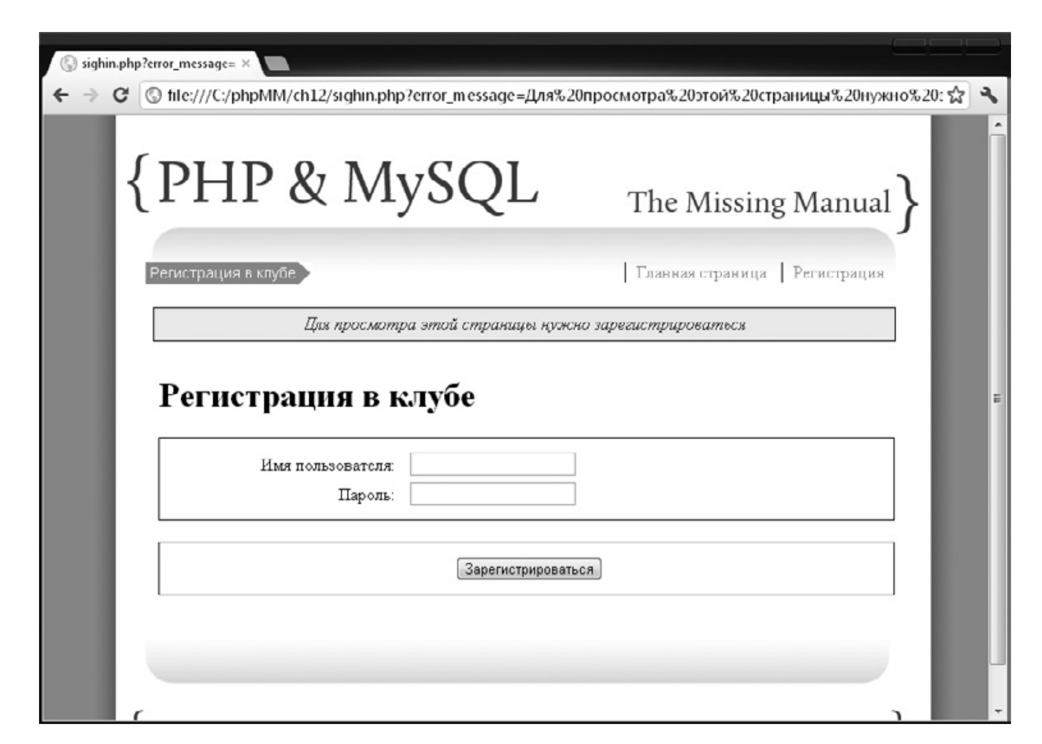


Рис. 12.14. Заблокированный доступ, полезное сообщение об ошибке и предоставление непосредственной возможности зарегистрироваться — именно этого вы и добивались

Итак, что еще осталось сделать? Посмотрим на исходный список.

- Нужен более качественный экран регистрации. Примитивное серое окно не нравится никому, пользователям хочется фирменной, стильной формы регистрации. **(Готово!)**
- Необходимо более четкое уведомление о том, зарегистрировался пользователь в приложении или нет. **(Готово!)**
- Нужен способ отмены регистрации. **(Готово!)**
- Необходимы два уровня аутентификации: один, чтобы попасть в основную часть приложения, и еще один административный уровень, чтобы попасть на такие страницы, как `show_users.php` и `delete_user.php`. **(Пока в этом плане ничего не сделано.)**
- Нужна какая-нибудь стандартная навигация, которая должна меняться на основе регистрации пользователя и той группы, к которой он принадлежит. **(Кое-что сделано.)**

Ну что ж, неплохо. Итак, немного передохните и приступайте к домашнему заданию: выполнению аутентификации на основе принадлежности к группе и выяснению причин, по которым cookie-файлы, может быть, и хороши, но не являются окончательным средством обеспечения аутентификации.

13 Авторизация и сессии

Осталось решить две серьезные задачи. Точнее полторы, поскольку часть этой работы уже сделана:

- нужны два уровня аутентификации: один, чтобы попасть в основную часть приложения, и еще один административный уровень, чтобы попасть на такие страницы, как `show_users.php` и `delete_user.php`;
- необходима какая-нибудь стандартная навигация, которая должна меняться на основе регистрации пользователя и той группы, к которой он принадлежит.

Что касается первого пункта, то у вас уже есть общая аутентификация, управляемая посредством сценария `authorize.php`. Но теперь нужно пойти дальше: необходимо улучшить `authorize.php`. Он должен разбираться в группе (или, точнее, в списке групп) и разрешать доступ только в том случае, если пользователь входит в те группы, имена которых ему переданы.

По второму пункту также проделана определенная работа. У вас есть меню, изменяющееся в зависимости от того, зарегистрировался пользователь или нет. Здесь также нужны улучшения: если пользователь входит в определенные группы, он должен иметь возможность администрирования записей пользователей, а также необходимо получить ссылку на страницу сценария `show_users.php` (в дополнение к стандартной ссылке на страницу сценария `show_user.php`).

Кроме того, существует проблема, связанная с cookie-файлами. Она, конечно, не столь значительна, но все же есть вполне обоснованная обеспокоенность насчет профессионального приложения, использующего для аутентификации одни лишь cookie-файлы. Следовательно, здесь есть чем заняться.

Моделирование групп в базе данных

Сначала о первостепенном. Перед тем как искать группы, к которым принадлежит пользователь, нужно получить эти группы в базе данных. А это, безусловно, означает, что вам нужны таблица для хранения групп и некие средства, с помощью которых пользователя можно будет связать с группой. Но это еще не все. Необходима возможность связать одного пользователя с несколькими группами.

В этой работе есть несколько четко выделяющихся этапов.

1. Создание таблицы в базе данных для хранения групп.
2. Отображение пользователя в никакой, одной или нескольких группах.
3. Создание кода PHP для поиска этого отображения.
4. Ограничение доступа к страницам на основе любой регистрации или конкретного набора групп.

И здесь все начинается с таблицы базы данных.

Добавление таблицы groups

Создание новой таблицы для программиста, работающего на PHP и MySQL, — дело вполне заурядное. Вы легко можете создать новую таблицу, дать ей имя, предоставить MySQL несколько столбцов, указать, у каких столбцов будет свойство, не допускающее нулевого значения — NOT NULL, и на этом создание таблицы будет довольно быстро завершено:

```
mysql> CREATE TABLE groups (
    ->   id          INT         NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->   name        VARCHAR(30) NOT NULL,
    ->   description VARCHAR(200)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> DESCRIBE groups;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id          | int(11)      | NO   | PRI | NULL    | auto_increment |
| name        | varchar(30)  | NO   |     | NULL    |                |
| description | varchar(200) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
3 rows in set (0.03 sec)
```

Как обычно, каждой группе необходим идентификатор и имя. Столбец description является необязательным, у него нет свойства NOT NULL. Вот и все, что вам нужно.

Без реальной информации о группах тестирование проводить довольно трудно, поэтому нужно добавить несколько групп в новую таблицу groups:

```
mysql> INSERT INTO groups
    ->        (name, description)
    -> VALUES ("Administrators",
    ->         "Пользователи-администраторы всего приложения.");
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO groups
```

```
    ->       (name, description)
    -> VALUES ("Luthiers",
    ->        "Изготовители гитар. Создают музыкальные инструменты.");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO groups
    ->       (name, description)
    -> VALUES ("Musicians",
    ->        "Говорят, что играют то, что вы чувствуете. И они чувствуют
               это.");
Query OK, 1 row affected (0.00 sec)
```

ПРИМЕЧАНИЕ

Создайте для своих пользователей те группы, которые считаете нужными. Главное, чтобы была создана группа Administrators. Если вы назовете эту группу как-нибудь по-другому, то вам нужно будет изменить все последующие ссылки, приводимые в данной главе с Administrators на то имя, которое вы будете использовать.

Как обычно, прежде чем пойти дальше, проверьте сделанное:

```
mysql> SELECT id, name FROM groups;
+----+----------------+
| id | name           |
+----+----------------+
|  1 | Administrators |
|  2 | Luthiers       |
|  3 | Musicians      |
+----+----------------+
3 rows in set (0.01 sec)
```

Отношение «многие ко многим»

Как связать пользователей с группами? Перед тем как приступить к этому на языке SQL, вы должны четко себе представить, как будут связаны между собой эти две таблицы. Отношения помогут вам определить, каким образом таблицы связаны друг с другом.

Один к одному, один ко многим, многие ко многим

Пример отношения «один к одному» вы уже видели. Когда вы сохраняли реальные изображения в базе данных, у вас была одна запись в таблице `users`, связанная с одной записью в таблице `images`. Следовательно, между таблицами `users` и `images` было отношение «один к одному».

Но это отношение для групп не подходит. Вы уже видели, что один пользователь может вообще не входить ни в какие группы, или входить в одну группу, или входить в несколько групп. Например, Майкл Гринфилд может входить в группы `Luthier`, `Musician` и `Administrator`. А может быть другой пользователь, не входящий ни в одну из этих групп.

Исходя из этого, вы имеете дело с отношением «один ко многим». Один пользователь может относиться к нескольким группам. Слово «многие» здесь не имеет буквального значения. Оно, скорее, означает «столько, сколько нужно». Следовательно, «многие» может означать 0, 1, 1000 или любое другое промежуточное количество.

Но это только часть истории. Вы можете повернуть все в обратную сторону и взглянуть на все это с позиции групп. В группе может быть много пользователей. Группа `Administrators` может иметь, к примеру, 4, 5 или 12 пользователей. Следовательно, если посмотреть на все со стороны отношения групп к пользователям, это отношение можно определить как «один ко многим», точно таким же отношением характеризуется и связь со стороны пользователей с группами.

Имеющееся здесь отношение между пользователями и группами (или, если вам так больше нравится, между группами и пользователями) можно определить как «многие ко многим». Один пользователь может входить в несколько групп, и у одной группы может быть несколько пользователей. Это многостороннее отношение, которое немного труднее смоделировать на уровне базы данных. Но в реальном мире данных оно играет не менее важную роль, чем отношения «один к одному» или «один ко многим».

ПРИМЕЧАНИЕ

Хорошим примером отношения «один ко многим» может послужить пользователь, у которого может быть целая галерея изображений. Пользователь может иметь несколько изображений, но эти пользовательские изображения не могут быть связаны с несколькими пользователями. Это по-настоящему можно считать отношением один (пользователь) ко многим (изображениям).

КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Многие программисты втайне любят математику

И это правда: у большинства программистов имеется хотя бы небольшая симпатия к математике, скрытая, зачастую, где-то в глубине души. Одним из подтверждений этому может послужить то, что многие концепции программирования берут идеи тех или иных понятий из математики.

Вы могли слышать об отношениях «один к одному» (1-к-1 или иногда 1:1). И вы также могли слышать об отношениях «один ко многим». Но также часто можно услышать об отношении 1-к-*N*. *N* — это математический термин, который обычно в математических записях приводится в виде буквы нижнего регистра *n*, а в программировании чаще всего обозначается заглавной буквой *N*. И эта *N* означает просто переменное число. Следовательно, *N* может быть нулем, единицей или каким-нибудь очень большим числом.

Значит, отношение «один ко многим» — это то же самое, что и отношение 1:*N*. То есть 1:*N* является более кратким способом сказать то же самое. Теперь вы уже знаете, что программисты склонны к сокращениям и лаконизмам. Поэтому на схемах баз данных можно часто увидеть обозначение 1:*N*, которое просто говорит вам о наличии отношения «один ко многим», имеющим место между двумя таблицами.

> А затем, конечно же, у вас получается отношение *N*:*N*, которое просто свидетельствует о том, что многие записи из одной таблицы имеют отношение ко многим записям из другой таблицы. Тем не менее отношение *N*:*N* (и отношение «многие ко многим», которое им представлено) на самом деле является концептуальной или виртуальной идеей. Как вы увидите далее, в большинстве систем для моделирования отношения *N*:*N* приходится на уровне базы данных создавать два отношения.

Объединение лучше всего осуществлять с помощью идентификаторов

Когда вы связывали пользователя с изображением профиля, вы применяли идентификатор (ID). У изображения имелся свой собственный ID, уникальным образом идентифицировавший эту картинку. Но у изображения также имелся идентификатор пользователя — user_id, с помощью которого осуществлялась связь картинки с конкретным пользователем в таблице users. Это упрощало получение изображения для пользователя при задействовании примерно следующего кода:

```
SELECT *
  FROM images
 WHERE user_id = $user_id;
```

Или вы могли объединить две таблицы таким образом:

```
SELECT u.username, u.first_name, u.last_name, i.filename, i.image_data
  FROM users u, images i
 WHERE u.id = i.user_id;
```

В обоих случаях в качестве объединителей использовались идентификаторы. Подобная схема хорошо работает в отношениях «один к одному» и «один ко многим». К стороне «многие» просто добавляется столбец, ссылающийся на идентификатор стороны «один». Таким образом сразу несколько изображений имеют столбец user_id, ссылающийся на пользователя с идентификатором 51 (или 2931, или каким-нибудь другим идентификатором, имеющимся в таблице users).

Но в случае с таблицами users и groups у нас нет отношений «один к одному» или «один ко многим». Как справиться с такой ситуацией?

Использование объединительной таблицы для связи users с groups

Идентификаторы (ID) являются ключами, с помощью которых довольно легко смоделировать отношение «один ко многим». А вот смоделировать отношения «многие ко многим» намного сложнее, поскольку установить связь по ID не удается. Нужна некая матрица: набор связанных между собой ID пользователей и ID групп.

Разберитесь с отношением «многие ко многим». В своей наипростейшей форме это фактически два отношения «один ко многим», с помощью которых можно прийти к выводу, что таблицы users и groups связаны друг с другом отношением «многие ко многим». Начните с одной стороны: с таблицы пользователей users. И определите, что здесь имеет место отношение «один ко многим». Затем посмотрите с другой стороны: с таблицы групп groups. Здесь также имеется отношение «один ко многим».

Таким же образом вы можете сконструировать отношение «многие ко многим» на уровне базы данных. У вас есть такая таблица, как users, которая связана с промежуточной таблицей. Назовите эту промежуточную таблицу user_groups и представьте, что у нее есть столбец user_id и столбец group_id. Следовательно, поле user_id может появиться в двух строках: в одной строке вместе с ID группы Administrators и еще в одной с ID группы Musicians. Тем самым вы получаете отношение «один ко многим» от таблицы users к таблице groups.

А затем вы также получаете отношение «один ко многим» от таблицы groups к таблице users. ID группы Administrators может появиться в таблице user_groups в пяти разных строках, по одному разу для каждого из пяти пользователей, входящих в эту группу.

Создайте данную таблицу и придайте этому замыслу конкретную форму:

```
mysql> CREATE TABLE user_groups (
    ->        user_id  INT NOT NULL,
    ->        group_id INT NOT NULL
    -> );
Query OK, 0 rows affected (0.03 sec)
```

Эта таблица становится мостом: каждая строка связывает одного пользователя с одной группой. Таким образом, для пользователя по имени Джефф Трюдо, имеющего идентификатор, равный 51, и входящего в группу Luthiers, которая имеет идентификатор, равный 2, в таблицу user_groups будет добавлена следующая строка:

```
mysql> INSERT INTO user_groups
    ->        (user_id, group_id)
    -> VALUES (51, 2);
Query OK, 1 row affected (0.02 sec)

mysql> select * from user_groups;
+---------+----------+
| user_id | group_id |
+---------+----------+
|      51 |        2 |
+---------+----------+
1 row in set (0.00 sec)
```

Сами по себе таблицы users и groups ничем не связаны. Но дополнительная таблица устанавливает между ними отношение «многие ко многим».

Проведение теста на принадлежность к группе

Чтобы увидеть принадлежность пользователя к группе, нужно посмотреть, есть ли в таблице user_groups запись, в которой содержится ID как желаемого пользователя, так и желаемой группы:

```
mysql> SELECT COUNT(*)
    ->   FROM users u, groups g, user_groups ug
    ->  WHERE u.username = "traugott"
    ->    AND g.name = "Luthiers"
    ->    AND u.user_id = ug.user_id
    ->    AND g.id = ug.group_id;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
```

В точку! Просмотрите не спеша этот запрос, который на первый взгляд может показаться немного сложным, но на самом деле он очень простой.

Сначала в нем можно использовать выражение COUNT(*), чтобы получить значение счетчика строк, возвращенных запросом. А затем нужно задействовать три таблицы: users, groups и user_groups.

```
SELECT COUNT(*)
  FROM users u, groups g, user_groups ug
```

После этого необходимо указать имя требуемого пользователя (используя на выбор любой столбец с именем, фамилией или названием пользователя), а также имя требуемой группы. Это приведет к выбору всего одной строки (или ни одной строки при отсутствии совпадения) в обеих таблицах users и groups:

```
SELECT COUNT(*)
  FROM users u, groups g, user_groups ug
 WHERE u.username = "traugott"
   AND g.name = "Luthiers"
```

Теперь нужно связать эти отдельные строки (каждую с использованием ее ID) с таблицей user_groups. Это самое простое объединение, в котором используются ID из каждой таблицы и проверяется их соответствие столбцам идентификаторов в таблице user_groups:

```
SELECT COUNT(*)
  FROM users u, groups g, user_groups ug
 WHERE u.username = "traugott"
   AND g.name = "Luthiers"
   AND u.user_id = ug.user_id
   AND g.id = ug.group_id;
```

Следовательно, это связывает нуль строк или одну строку в таблице users с таблицей user_groups, которая также связана с нулем строк или с одной строкой в таблице groups. Что получается в результате? Либо одна-единственная строка со значением COUNT, равным 1, что означает наличие связи пользователя из таблицы users с указанной вами группой:

```
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
```

Либо строка со значением COUNT, равным 0, что означает отсутствие связи:

```
mysql> SELECT COUNT(*)
    ->   FROM users u, groups g, user_groups ug
    ->  WHERE u.username = "traugott"
    ->    AND g.name = "Administrators"
    ->    AND u.user_id = ug.user_id
    ->    AND g.id = ug.group_id;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.05 sec)
```

ВНИМАНИЕ

Осторожно! Применяя конкретно это выражение, в котором используется COUNT, вы всякий раз будете получать одну строку. Важной информацией является значение, имеющееся в строке, а не наличие самой строки.

Теперь нужно только превратить все это в код PHP.

Проверка на принадлежность к группе

Вы уже имеете опыт создания неплохой схемы аутентификации. Вы заменили стандартную аутентификацию собственной схемой аутентификации, которая заключается в том, что вы позволяете пользователю войти в приложение, если он зарегистрировался. Пользователь проходит аутентификацию таким образом, что ваша система понимает, что пользователь является именно тем, кем он назвался.

Но теперь настало время добавить авторизацию: возможность получения доступа только к конкретным страницам на основе более определенного признака. В самом простом варианте вам нужен некий уровень авторизации, проводимой с помощью сценария authorize.php: авторизацию проходят только те пользователи, которые прошли аутентификацию. Но обычно авторизация идет намного дальше этого варианта. Она имеет более выборочный характер, управляя доступом на основе, скажем, принадлежности к группе.

На данный момент у вас есть пользователи и группы, а также связь между двумя данными категориями. Следовательно, теперь сценарий authorize.php должен быть усовершенствован для работы с этими группами в вашей схеме авторизации.

Сценарий authorize.php нуждается в функции

Сейчас сценарий authorize.php запускается автоматически при вызове из какого-нибудь другого сценария. Следовательно, код в сценарии authorize.php не оформлен в виде функции, он просто находится в теле PHP-файла:

```
<?php

if ((!isset($_COOKIE['user_id'])) || (!strlen($_COOKIE['user_id']) > 0)) {
  header('Location: signin.php?' .
    'error_message=Для просмотра этой страницы нужно зарегистрироваться.');
          //
  exit;
}
?>
```

Пока этот код работал нормально. Но времена меняются. Вам нужны средства, с помощью которых вы могли бы передать группу или список групп сценарию authorize.php, чтобы затем этот сценарий мог осуществить перебор этих групп и узнать, связан ли с ними текущий пользователь. Такая ситуация, когда нужен блок кода, который должен принять часть информации, просто вопит о необходимости создания функции. Есть и другие варианты, но их сложнее объяснить и обслужить. (Если эти варианты вас интересуют, обратитесь к следующей врезке.)

Создайте в сценарии authorize.php новую функцию. В конечном счете она должна получать массив групп, разрешающий доступ. На данный момент вы можете установить значение по умолчанию для аргумента функции и использовать это значение для поддержки текущей функциональности: разрешить доступ любому авторизировавшемуся пользователю:

```
<?php

function authorize_user($groups = NULL) {
  // Если не создан cookie-файл, проверять группы не нужно
  if ((!isset($_COOKIE['user_id'])) ||
      (!strlen($_COOKIE['user_id']) > 0)) {
    header('Location: signin.php?' .
      'error_message= Для просмотра этой страницы нужно зарегистрироваться.');
    exit;
  }
}
?>
```

Теперь вернитесь к сценарию show_user.php и добавьте явный вызов этой функции. Ей не нужно передавать какие-либо имена групп. Страница сценария show_user.php должна быть открыта для любого зарегистрировавшегося пользователя:

```
<?php

require '../scripts/authorize.php';
require '../scripts/database_connection.php';
require '../scripts/view.php';

// Авторизация любого зарегистрировавшегося пользователя
authorize_user();

// Получение ID пользователя, чью информацию нужно показать
$user_id = $_REQUEST['user_id'];

// Создание инструкции SELECT

// и т. д...
```

Этот код нужно протестировать. Поскольку предоставляемые по умолчанию функциональные возможности должны быть теми, которые у вас уже есть, следует удостовериться, что вы не можете получить доступ к странице сценария show_user.php без предварительной регистрации. Введите URL в адресную строку браузера. Вы должны увидеть свою страницу регистрации, показанную на рис. 13.1.

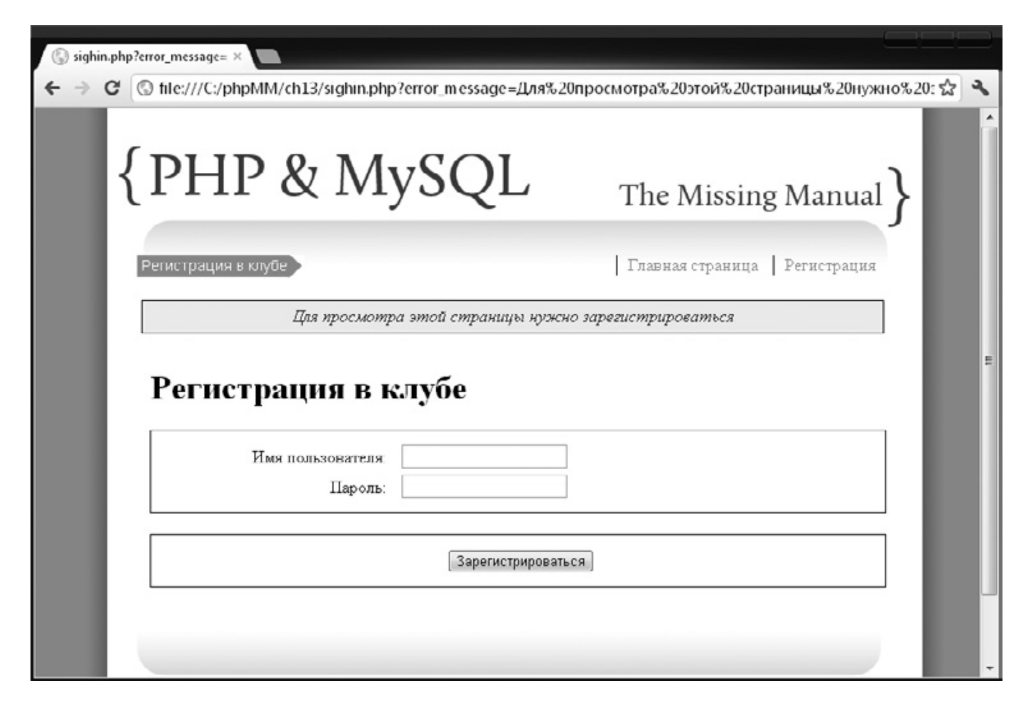


Рис. 13.1. Тестирование нововведений нужно начинать с тестирования старых возможностей

Получение списка групп

Настало время приступить к сути всей этой работы. Начните с отправки в адрес функции authorize_user списка групп посредством массива PHP. Это можно сделать в сценариях show_users.php и delete_user.php, доступ к странице каждого из которых требует принадлежности пользователя к группе Administrators.

```
<?php

require_once '../scripts/app_config.php';
require_once '../scripts/authorize.php';
require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

// Получить доступ к этой странице могут только пользователи из группы
// Administrators
authorize_user(array("Administrators"));

// Остальной код PHP и HTML-вывод
```

ПРИМЕЧАНИЕ

Изменения, показанные выше, относятся к сценарию show_users.php. Внесите такие же изменения в сценарий delete_user.php, чтобы к нему нельзя было получить непосредственный доступ.

ЗАМЕТКИ О ПРОЕКТИРОВАНИИ

С функциями и без функций

В сценарии authorize.php есть функция, получающая или не получающая группы через аргумент. Но это только один из способов решения вопроса. Есть и другие подходы: вы можете, к примеру, создать переменную, а затем использовать ее в требуемом файле.

Возьмем простой сценарий:

```
<?php
$message = "hello\n\n";
require_once "print.php";
?>
```

Если хотите, его можно назвать test.php. А затем предположим, что сценарий print.php, на который имеется ссылка, выглядит следующим образом:

```
<?php
echo $message;
?>
```

Когда осуществляется запрос сценария print.php, это похоже на вставку кода, имеющегося в print.php, именно в то место, где находится строка с инструкцией

➩

`require_once`. Это означает, что при запуске этого сценария код PHP приобретает в конечном итоге следующий вид:

```
<?php
$message = "hello\n\n";
echo $message;
?>
```

Запустите `test.php` и получите следующий результат:

```
yellowta@yellowtagmedia.com [~/www/phpMM/ch13]# php test.php
X-Powered-By: PHP/5.2.17
Content-type: text/html
hello
```

Следовательно, можно «передать» информацию в нужный сценарий и таким образом.

Этот подход невыгоден не потому, что его трудно реализовать или есть какие-то вопросы с его реализацией. Проблема в том, что он не столь очевиден.

Вот как будет выглядеть код для авторизации:

```
$allowed_groups = array("Musicians", "Luthiers");
require_once "../scripts/authorize.php";
```

Здесь все правильно. Просто в нем непонятно, что переменная `$allowed_groups` нужна до инструкции `require_once`, применяемой в отношении сценария `authorize.php`, и что `authorize.php` пользуется этой переменной. Поэтому пусть функция `authorize_user` выглядит несколько неуклюже. С ней все понятнее и лучше, чем с ее альтернативой, то есть с кодом, который труднее читать, пока вы заранее не узнаете, что он делает.

Массив является, пожалуй, чуть ли не самым простым средством в PHP для предоставления списка какой-нибудь функции. Сейчас в сценарии `authorize.php` вы либо ничего не получаете, либо получаете список имен разрешенных групп. Следовательно, можно приступать к работе с этими группами.

Но сначала нужно заставить функцию `authorize_user` возвращать управление, если переданный ей аргумент имеет либо значение пустого списка, либо `NULL`. В этих двух случаях вести поиск в базе данных нет никакой необходимости.

```
<?php

function authorize_user($groups = NULL) {
  // Если не создан cookie-файл, проверять группы не нужно
  if ((!isset($_COOKIE['user_id'])) ||
      (!strlen($_COOKIE['user_id']) > 0)) {
    header('Location: signin.php?' .
     'error_message= Для просмотра этой страницы нужно зарегистрироваться.');
    exit;
  }

  // Если группы не были переданы, достаточно и этой авторизации
```

```
  if ((is_null($groups)) || (empty($groups))) {
    return;
  }
}
?>
```

ПРИМЕЧАНИЕ

Функции empty передается практически любой тип данных PHP. Она определяет, что в данном случае подходит под термин «пустой» (empty), и возвращает либо true, либо false. Что касается массива, функция empty возвращает true, если в массиве нет элементов.

Инструкция return заставляет интерпретатор PHP вернуть управление сценарию, из которого был сделан вызов. Это позволяет сценарию продолжить выполнение, то есть дать пользователю возможность увидеть запрошенную страницу.

Последовательный перебор групп

Теперь вернемся к ситуации, когда вы получаете список групп, которая происходит в сценариях show_users.php и delete_user.php. В таких случаях сценарий authorize.php должен осуществить последовательный перебор групп и для каждой группы создать SQL-запрос.

Начните с циклического перебора элементов массива $groups. Можно использовать цикл for, но в данном случае есть более подходящий выбор: foreach. Цикл foreach позволяет осуществить последовательный перебор элементов массива и автоматически присваивать переменной значение текущего элемента этого массива:

```
$my_array = array("first", "second", "third");
foreach ($my_array as $item) {
  echo $item;
}
```

Следовательно, для массива $groups можно применить следующий цикл:

```
foreach ($groups as $group) {
  // осуществление SQL-поиска для текущего значения переменной $group
}
```

Теперь подумайте о том, что происходит внутри цикла. Вам нужно что-либо подобное исходному SQL, использовавшемуся для связи таблицы users с таблицей groups:

```
SELECT COUNT(*)
  FROM users u, groups g, user_groups ug
 WHERE u.username = "traugott"
   AND g.name = "Luthiers"
   AND u.user_id = ug.user_id
   AND g.id = ug.group_id;
```

Но этот код сложнее необходимого для сценария authorize.php. Начнем с того, что вам здесь вообще не нужна таблица users. Эта таблица — только часть запроса,

нужного для связи имени пользователя с его идентификатором user_id. Но у вашего приложения уже есть идентификатор user_id. Поэтому все упрощается и приобретает следующий вид:

```
SELECT COUNT(*)
  FROM user_groups ug, groups g
 WHERE g.name = mysql_real_escape_string($group)
   AND g.id = ug.group_id
   AND ug.user_id = mysql_real_escape_string($_COOKIE['user_id']);
```

ПРИМЕЧАНИЕ

Как обычно, нужно воспользоваться функцией mysql_real_escape_string, чтобы обеспечить получение базой данных чистых значений. Было бы неплохо выработать привычку использовать mysql_real_escape_string в отношении практически любых данных, появляющихся в ваших сценариях и отправляемых базе данных MySQL.

Можно внести и еще одно усовершенствование. В показанном выше запросе вам не нужна получаемая в результате его выполнения строка и не нужно выяснять, какое у этой строки значение: 0 (не входит в группу) или 1 (входит в группу). Это дополнительный шаг. Лучше просто проверить, есть ли вообще какой-нибудь результат. Иными словами, вам нужен запрос, который при наличии совпадения просто возвращает строку. Следовательно, можно внести еще одно изменение:

```
SELECT ug.user_id
  FROM user_groups ug, groups g
 WHERE g.name = mysql_real_escape_string($group)
   AND g.id = ug.group_id
   AND ug.user_id = mysql_real_escape_string($_COOKIE['user_id']);
```

Какой именно столбец выбрать из таблицы user_groups, не имеет значения, подойдет и ug.group_id. Здесь главное, что вы получите какой-нибудь результат при наличии соответствия или не получите никакого результата. Таким образом, код сделает на один шаг меньше.

Соберите все это вместе, и в цикле foreach у вас получится примерно следующее:

```
foreach ($groups as $group) {
  // осуществление SQL-поиска для текущего значения переменной $group
  $query = "SELECT ug.user_id" .
           "  FROM user_groups ug, groups g" .
           " WHERE g.name = '" . mysql_real_escape_string($group) . "'" .
           "   AND g.id = ug.group_id" .
           "   AND ug.user_id = " .
               mysql_real_escape_string($COOKIE['user_id']) . "';";
  mysql_query($query);

  // Обработка результатов
}
```

Этот сценарий работает и без запроса таблицы users. Но данная строка будет создаваться снова и снова. Для каждой группы эта строка будет создаваться заново, а это слишком расточительно.

И здесь вам на помощь опять придет функция sprintf. Используя ее, вы можете создать одну строку, дать ей один или два символа подмены и просто вставлять в строку значения для каждого символа подмены. Строку можно считать неизменной, а вы просто изменяете в строке данные, которые имеют непостоянный характер.

Таким образом можно будет создать строку запроса за пределами цикла foreach:

```
// Создание строки запроса
$query_string =
    "SELECT ug.user_id" .
    "  FROM user_groups ug, groups g" .
    " WHERE g.name = '%s'" .
    "   AND g.id = ug.group_id" .
    "   AND ug.user_id = " . mysql_real_escape_string($_COOKIE['user_id']);

foreach ($groups as $group) {
  // осуществление SQL-поиска для текущего значения переменной $group

  // Обработка результатов
}
```

Затем внутри цикла foreach нужно воспользоваться функцией sprintf для предоставления значений, попадающих в строку для конкретной группы:

```
// Создание строки запроса
$query_string =
    "SELECT ug.user_id" .
    "  FROM user_groups ug, groups g" .
    " WHERE g.name = '%s'" .
    "   AND g.id = ug.group_id" .
    "   AND ug.user_id = " . mysql_real_escape_string($_COOKIE['user_id']);

foreach ($groups as $group) {
  // SQL-поиск для текущего значения переменной $group
  $query = sprintf($query_string, mysql_real_escape_string($group));
  $result = mysql_query($query);

  // Обработка результатов
}
```

Заметьте, что вдобавок к использованию sprintf этот код передает идентификатор текущего пользователя, извлеченный из переменной $_COOKIE, строке, которая собирается за пределами цикла. Передавать ее функции sprintf не нужно, потому что она не будет изменяться при проходе цикла.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

А что, разве без данных изменений все эти запросы не будут работать?

Конечно, будут. Но, как вы уже поняли, из всего разнообразия решений проблемы можно выделить более удачные. При работе с базами данных «лучше» обычно означает «быстрее», а «быстрее» означает, что «базе данных достается меньше работы».

В случае поиска группы и определения принадлежности пользователя к этой группе функционально правильным можно считать следующий запрос:

```
SELECT COUNT(*)
  FROM users u, groups g, user_groups ug
 WHERE u.username =
       mysql_real_escape_string($_COOKIE['username'])
   AND g.name = mysql_real_escape_string($group)
   AND u.user_id = ug.user_id
   AND g.id = ug.group_id;
```

Но здесь объем работы существенно превышает минимально необходимый. Задействуется лишняя таблица (users), от чего можно избавиться, поскольку идентификатор пользователя уже имеется в cookie-файле.

Можно сократить код и с точки зрения получения результатов, если отойти от использования ключевого слова COUNT в инструкции SELECT, которое всегда требует проверять результаты в строке в зависимости от содержимого столбца в таблице user_groups. А вам нужно только посмотреть, есть ли вообще какие-нибудь возвращенные строки. При этом значения в этих строках уже не представляют никакого интереса.

Кроме того, сократить общее время выполнения можно за счет однократного создания строки и использования функции sprintf для модификации только небольшой части этой строки при каждом переходе к новой группе. Пусть это и не самое значительное усовершенствование, но все же его можно отнести к числу важных и легкодоступных.

Все эти небольшие изменения могут складываться в заметное улучшение работы вашего приложения. У него просто сократится время реакции. Важность всего этого еще более возрастет, когда появится необходимость запуска сценария авторизации при каждом обращении пользователя к вашей странице. Медленно работающий и перегруженный ненужным кодом сценарий станет замедлять работу при каждом доступе к странице.

Большинству пользователей не нравятся медленно загружающиеся сайты, а многие просто не хотят на них заходить. Речь здесь идет не о паузе при решении вопросов безопасности доступа или при просмотре доставленной информации, а о переходе к новой странице. Небольшие усилия по оживлению работы сайта существенно улучшают впечатления пользователей от его работы, особенно если количество пользователей, посещающих сайт, постоянно увеличивается, что сопровождается возрастающим количеством обращений к вашей базе данных для проверки их принадлежности к той или иной группе.

Разрешить, отказать, перенаправить

Когда есть надежный запрос, можно перейти к обработке его результатов. Можете проверить количество строк и узнать все, что вам нужно: если ни одна строка не возвращена, значит, пользователь не входит в группу, показанную в переменной $group, и цикл должен продолжиться, перейдя к следующему значению переменной $group, взятому из массива $groups.

Если есть строка, возвращенная в результате запроса, то это значит не только, что пользователь входит в разрешенную группу, но и то, что работу authorize_user нужно остановить. В продолжении циклического перебора элементов массива $groups больше нет необходимости, нужно просто вернуть управление сценарию, из которого была вызвана эта функция, позволив выполняться коду PHP и HTML, имеющемуся в этом сценарии.

И последний случай: все группы были проверены, и ни для одной из них не была возвращена строка. В этом случае цикл foreach завершает свою работу, и возвращать управление вызывавшему функцию сценарию не нужно, поскольку это позволит пользователю «войти», а это противоречит тому, что должно произойти. Не подходит также и перенаправление пользователя на страницу регистрации. По крайней мере в большинстве случаев он уже зарегистрировался и просто не имеет прав доступа к текущей странице.

Что тогда остается делать? В самом простом случае еще раз воспользоваться функцией обработки ошибки handle_error. Но вы можете разобраться с ситуацией самостоятельно. Наверное, можно перенаправить пользователя на последнюю страницу, которую он до это просматривал, и отправить ему сообщение об ошибке. Или создать собственную страницу, дающую пользователю возможность запросить права на какую-нибудь другую страницу. Неважно, как вы со всем этим разберетесь, в любом случае вы пошлете пользователю что-то другое, не показывая текущую страницу.

Версия authorize.php, в которой учтены все эти случаи, имеет следующий вид:

```
<?php

require_once 'database_connection.php';
require_once 'app_config.php';

function authorize_user($groups = NULL) {

  // Если не создан cookie-файл, проверять группы не нужно
  if ((!isset($_COOKIE['user_id'])) || (!strlen($_COOKIE['user_id']) > 0)) {
    header('Location: signin.php?' .
     'error_message= Для просмотра этой страницы нужно зарегистрироваться.');
    exit;
  }

  // Если группы не были переданы, достаточно этой авторизации
```

```
  if ((is_null($groups)) || (empty($groups))) {
    return;
  }

  // создание строки запроса
  $query_string =
      "SELECT ug.user_id" .
      "  FROM user_groups ug, groups g" .
      " WHERE g.name = '%s'" .
      "   AND g.id = ug.group_id" .
      "   AND ug.user_id = " . mysql_real_escape_string($_COOKIE['user_id']);


  // Перебор всех групп и проверка принадлежности к каждой из них
  foreach ($groups as $group) {
    $query = sprintf($query_string, mysql_real_escape_string($group));
    $result = mysql_query($query);

    if (mysql_num_rows($result) == 1) {
      // Если получен результат, пользователю нужно разрешить доступ.
      // Возвращение управления, чтобы продолжилось выполнение сценария.
      return;
    }
  }

  // Если мы попали сюда, значит соответствий не было ни в одной из групп.
  // Доступ пользователю не разрешен.
  handle_error("Вы не прошли авторизацию для просмотра данной страницы.");
  exit;
}
?>
```

Прошло довольно много времени, и вы наконец-то можете все проверить в работе. Убедитесь в том, что у вас есть пользователь в таблице `users`, который входит в группу `Administrators` (с помощью таблицы `user_groups`), и есть еще один пользователь, не входящий в эту группу. Первый из них должен получить возможность перехода на страницу сценария `show_users.php` без каких-либо проблем, а второй должен быть отправлен на страницу ошибки, показанную на рис. 13.2.

Вы должны увидеть эту страницу в качестве первого, а не последнего шага в авторизации. Такая настройка выглядит несколько нелепо, и вам нужно найти какой-нибудь более подходящий, не прерывающий работу способ, который позволит пользователю понять, что его не пустили туда, где его не должно быть. По возможности пользователя нужно вернуть на страницу, к которой он имеет доступ. Для сообщения об ошибке, занимающего всю страницу, должен быть более веский повод, и она редко показывается без серьезных на то причин.



Рис. 13.2. Страница ошибки для пользователя, не прошедшего авторизацию

Меню, ориентированное на принадлежность к той или иной группе

Сейчас вы можете воспользоваться функцией `authorize_user` для проверки принадлежности пользователя к группе, входящей в список групп, и либо отклонить доступ к странице, либо позволить пользователю просматривать эту страницу. Это означает, что у вас есть логика для управления меню, которое ориентировано на принадлежность пользователя к той или иной группе, но для его реализации нужно кое-что переделать.

Посмотрите на вашу систему меню в ее текущем состоянии в сценарии `view.php`:

```
function display_title($title, $success_message = NULL, $error_message = NULL)
{
echo <<<EOD
<body>
<div id="page_start">
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">$title</div>
  <div id="menu">
    <ul>
      <li><a href="index.html">Главная страница</a></li>
```

```
EOD;
  if (isset($_COOKIE['user_id'])) {
    echo "<li><a href='show_user.php'>Профиль</a></li>";
    echo "<li><a href='signout.php'>Отмена регистрации</a></li>";
  } else {
    echo "<li><a href='signin.php'>Регистрация</a></li>";
  }
echo <<<EOD
    </ul>
  </div>
EOD;
  display_messages($success_message, $error_message);
  echo "</div> <!-- тег, закрывающий контейнер с id="page_start" -->";
}
```

Функция authorize_user не относится к функциям, которые можно было бы просто вставить в этот код. Она либо позволяет пользователю увидеть страницу, либо не позволяет ему сделать это. Это не какой-нибудь тонко настроенный инструмент, дающий возможность проверить принадлежность к той или иной группе и получить назад значение true или false.

В действительности вам нужен какой-нибудь такой код:

```
function display_title($title, $success_message = NULL, $error_message = NULL)
{
echo <<<EOD
 <body>
<div id="page_start">
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">$title</div>
  <div id="menu">
    <ul>
      <li><a href="index.html">Главная страница</a></li>
EOD;
  if (isset($_COOKIE['user_id'])) {
    echo "<li><a href='show_user.php'>Профиль</a></li>";
    if (user_in_group($_COOKIE['user_id'], "Administrators")) {
      echo "<li><a href='show_users.php'>Управление</a></li>";
    }
    echo "<li><a href='signout.php'>Отмена регистрации</a></li>";
  } else {
    echo "<li><a href='signin.php'>Регистрация</a></li>";
  }
echo <<<EOD
    </ul>
  </div>
EOD;
  display_messages($success_message, $error_message);
  echo "</div> <!-- тег, закрывающий контейнер с id="page_start -->";
}
```

ПРИМЕЧАНИЕ

Чтобы этот код заработал, нужно также в сценарий view.php добавить инструкцию require_once для authorize.php.

Затем эта функция проверит принадлежность к группе и покажет для тех, кто входит в группу `Administrators`, ссылку **Управление**. Весь соответствующий код в сценарии `authorize_user.php` уже есть:

```
// Создание строки запроса
$query_string =
    "SELECT ug.user_id" .
    "  FROM user_groups ug, groups g" .
    " WHERE g.name = '%s'" .
    "   AND g.id = ug.group_id" .
    "   AND ug.user_id = " . mysql_real_escape_string($_COOKIE['user_id']);


// Перебор всех групп и проверка принадлежности к каждой из них
foreach ($groups as $group) {
  $query = sprintf($query_string, mysql_real_escape_string($group));
  $result = mysql_query($query);

  if (mysql_num_rows($result) == 1) {
    // Если получен результат, пользователю нужно разрешить доступ.
    // Возвращение управления, чтобы продолжилось выполнение сценария.
    return;
  }
}
```

Этот код нужно только приспособить к новой функции, которой передается идентификатор пользователя и группа. Итак, все довольно просто:

```
function user_in_group($user_id, $group) {
  $query_string =
    "SELECT ug.user_id" .
    "  FROM user_groups ug, groups g" .
    " WHERE g.name = '%s'" .
    "   AND g.id = ug.group_id" .
    "   AND ug.user_id = %d";
  $query = sprintf($query_string, mysql_real_escape_string($group),
                                  mysql_real_escape_string($user_id));
  $result = mysql_query($query);

  if (mysql_num_rows($result) == 1) {
    return true;
  } else {
    return false;
  }
}
```

Здесь нет ничего нового. Это просто новый перепев старого хита: кода, который уже был в сценарии authorize.php в функции authorize_user.

Вставьте этот код и испытайте его в работе. Сначала зарегистрируйтесь от имени пользователя, не входящего в группу Administrators. Зайдите, к примеру, на страницу сценария show_user.php. В пунктах меню не должно быть ссылки на Управление (рис. 13.3).



Рис. 13.3. Пользователь не входит в группу Administrators, поэтому не видит пункт меню Управление

Теперь откажитесь от регистрации и проделайте все то же самое, но на этот раз от имени пользователя, входящего в группу администрации. Здесь волшебным образом, по крайней мере для программистов, не работающих на PHP, появится новый пункт меню. Ссылку на Управление можно увидеть на рис. 13.4.

С точки зрения администратора он просто получает дополнительную возможность. Управление открывается только потому, что эта возможность доступна. Но есть еще один повод для размышлений: группа Administrators фигурирует в нескольких местах. Можно подумать о создании константы или даже функции is_admin, чтобы не нужно было вспоминать, как пишется слово Administrators.



Рис. 13.4. Дополнительный пункт в меню для группы Administrators

Введение в практику использования сессий браузера

До сих пор секретом успеха большинства аутентификаций и авторизаций были cookie-файлы. Но многие программисты очень не любят решения, построенные на хранении пользовательской информации с помощью только лишь cookie-файлов. Самой большой проблемой cookie-файлов является то, что они находятся только на стороне клиентов. Следовательно, все, что вы сохраняете в cookie-файле, существует только на клиентском компьютере.

В вашем случае это означает, что идентификатор пользователя и его имя хранятся на вашем компьютере. По сути, в любом браузере не составит труда просмотреть содержимое cookie-файлов. Зарегистрируйтесь в своем приложении, а затем просмотрите свои cookie-файлы. Например, в Mozilla Firefox вы можете выполнить команду `Инструменты` ▸ `Веб-разработка` ▸ `Просмотр информации в cookie-файлах`. Cookie-файлы, связанные с данным приложением, показаны на рис. 13.5.

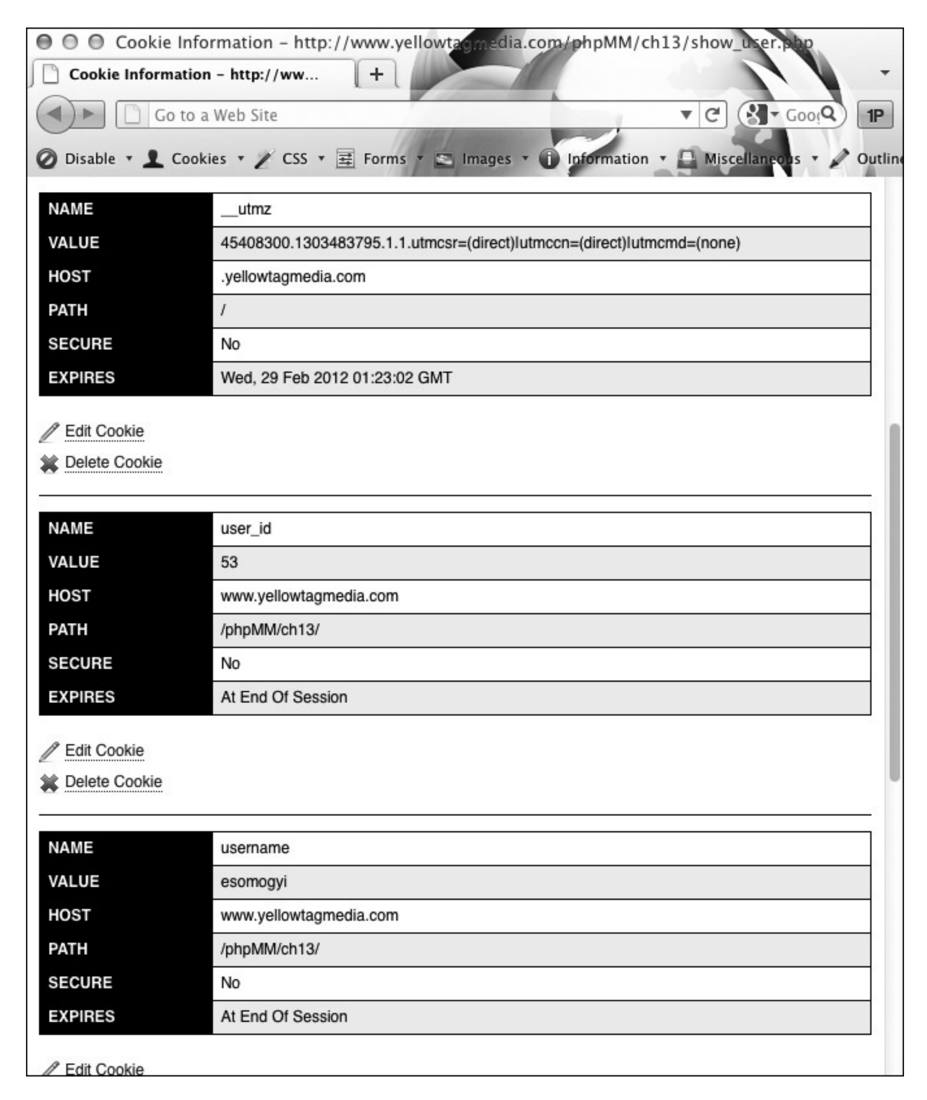


Рис. 13.5. Вы можете увидеть cookie-файлы user_id и username, а также некоторые другие, связанные с кодами, которые браузер использует для поддержки наряду с cookie-файлами вашего приложения

ПРИМЕЧАНИЕ

В браузере Safari cookie-файлы можно найти в меню Настройки. Щелкните на вкладке Конфиденциальность, а затем на кнопке Подробности. В браузере Google Chrome нужно выполнить команду Настройки ▸ Показать дополнительные настройки ▸ Настройки контента ▸ Все файлы cookie и данные сайтов. В браузере Internet Explorer необходимо выбрать Сервис ▸ Свойства обозревателя, щелкнуть на вкладке Общие и в области История просмотра нажать кнопку Параметры. Затем в области Временные файлы Интернета следует выбрать пункт Показать файлы. Все эти варианты дадут вам одну и ту же информацию, хотя в каждом случае она будет выглядеть немного по-разному.

ЧАСТО ЗАДАВАЕМЫЙ ВОПРОС

Должна ли функция authorize_user вызывать функцию user_in_group?

Если проанализировать суть переделок, то получится, что в функции `user_in_group` есть продублированный код, что не может не вызвать беспокойства. Действительно, код в функции `user_in_group` и код перебора элементов массива `$groups` и поиска в каждой группе в функции `authorize_user` очень похожи друг на друга.

Выгоду от существования функции `user_in_group` и от удаления одинакового кода можно получить путем переработки кода цикла `foreach` в функции `authorize_user`:

```
// Удалите исходную строку запроса перед циклом

// Пройдите каждую группу и проверьте принадлежность к ней
foreach ($groups as $group) {
  if (user_in_group($_COOKIE['user_id'],
$group) {
    // Просто верните управление, чтобы сценарий продолжил свое выполнение
    return;
  }
}
```

Выглядит неплохо. Стало намного меньше кода, и, похоже, переделка вполне удалась. Но, по сути, вы вернулись к исходному коду `authorize_user`, от которого уже отошли. Теперь получается, что строка запроса создается при каждом проходе цикла (хотя это и скрыто внутри функции `user_in_group`). Эта строка создается снова и снова, и ей постоянно присваивается один и тот же идентификатор пользователя с каждой группой, имеющейся в массиве `$groups`. Отойдя от такого подхода, вы (пусть и немного) повысили производительность работы функции `authorize_user`.

И здесь придется принимать непростое решение. Стоит ли понятный, требующий переделки подход потери скорости и некоторой продублированности кода? В данном случае, когда имеющая небольшой объем функция `authorize_user` потенциально вызывается для большинства, если не для каждой страницы, вряд ли стоит проводить переделку. Мы получаем небольшое увеличение скорости при сотне просмотров страницы, а если это будут тысячи, миллионы просмотров... этот прирост уже будет весьма существенным.

Хранилище на стороне клиента — главная причина того, что многие разработчики не любят cookie-файлы. Когда клиентский компьютер является публичной машиной в какой-нибудь библиотеке или домашней машиной, то небезопасно оставлять системное значение такого уровня, как идентификатор пользователя на любой старой машине.

И это весьма серьезный вопрос. Данный пользовательский ID уникальным образом идентифицирует пользователя в вашей базе данных. Кроме того, большинство приложений, использующих cookie-файлы, увеличивают объем дополнительной информации на машине клиента, вместо того чтобы его уменьшать. Путем хранения в cookie-файлах данных о пользовательских группах (или идентификаторов этих групп) можно ускорить поиски пользователей и групп. В cookie-файлах можно также хранить какую-нибудь персональную информацию, которую не хочется постоянно искать в базе данных.

И вся эта информация в конечном счете остается на компьютере вашего пользователя до тех пор, пока не истечет срок использования cookie-файлов. Так что же делать программисту, чтобы себя от этого обезопасить?

Сессии находятся на серверной стороне

Сессии по всеобщему признанию являются «ответом» на cookie-файлы. Сессии очень похожи на cookie-файлы тем, что в них может храниться информация. Но сессии имеют два серьезных отличия.

- Сессии хранятся на сервере, а не на клиенте. Данные сессий нельзя просматривать с помощью браузера. Там просто нечего смотреть за исключением, возможно, нечитаемого идентификатора, связывающего конкретный браузер с сессией.
- Поскольку сессии хранятся на сервере, они могут использоваться для хранения существенно более объемных фрагментов данных, чем cookie-файлы. К примеру, на сервере в сессии можно хранить изображение пользовательского профиля и не волноваться о расходе пространства на пользовательском компьютере.

Поскольку при использовании сессий потенциально конфиденциальная информация на пользовательском компьютере не хранится, многие программисты предпочитают работать с сессиями.

Сессии должны быть запущены

Самое большое изменение при работе с сессиями относится не к новому синтаксису. Фактически вы довольно быстро поймете, что изменения по сравнению с использованием cookie-файлов, весьма незначительные. Но есть одна существенная разница: перед работой с сессиями нужно вызвать функцию `session_start`:

```
// Начало (возобновление) сессий
session_start();

// А теперь работа с информацией сессии
```

Вполне естественно возникает мысль: вызвать функцию session_start в сценарии signin.php и можно идти дальше. А вот то самое место, где нужно впервые вызвать функцию session_start:

```
<?php

require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';

$error_message = $_REQUEST['error_message'];

session_start();

// Весь остальной код PHP и HTML...
```

Вызов session_start в этом месте запускает PHP-механизм, делающий сессии доступными.

От $_COOKIE к $_SESSION

Теперь станет полегче: вместо сверхглобальной переменной $_COOKIE нужно использовать суперглобальную переменную $_SESSION. Все просто:

```
<?php

require_once '../scripts/database_connection.php';
require_once '../scripts/view.php';
$error_message = $_REQUEST['error_message'];

session_start();

// Если пользователь зарегистрировался, в сессии будет создан
// элемент user_id
if (!isset($_SESSION['user_id'])) {
  // и т. д...
```

А затем нужно внести еще одно небольшое изменение. При работе с сессиями функция setcookie не используется. Вместо нее значения присваиваются непосредственно переменной $_SESSION, при этом предоставляются ключ и значение:

```
if (!isset($_SESSION['user_id'])) {

 // Установка факта отправки формы с именем пользователя для регистрации
 if (isset($_POST['username'])) {
    // Попытка зарегистрировать пользователя
    $username = mysql_real_escape_string(trim($_REQUEST['username']));
    $password = mysql_real_escape_string(trim($_REQUEST['password']));

    // Поиск пользователя
```

```
    $query = sprintf("SELECT user_id, username FROM users " .
                     " WHERE username = '%s' AND " .
                     "       password = '%s';",
                     $username, crypt($password, $username));

    $results = mysql_query($query);

    if (mysql_num_rows($results) == 1) {
      $result = mysql_fetch_array($results);
      $user_id = $result['user_id'];
      // Теперь больше нет функции setcookie
      $_SESSION['user_id'] = $user_id;
      $_SESSION['username'] = $username;
      header("Location: show_user.php");
      exit();
    } else {
      // Если пользователь не найден, выдача ошибки
      $error_message = "Вы дали неверную комбинацию имя пользователя-пароль.";

    }
  }
```

Итак, теперь для извлечения значений из сессии используется переменная `$_SESSION`, и она же применяется для вставки значений в сессию.

Все это время информация хранится на сервере, а не на клиентской машине. Правда же, неплохо?

Сессии должны быть еще и перезапущены

А теперь кое-что странное. Попробуйте войти в приложение, используя действующую комбинацию имени пользователя и пароля. Вы не увидите того, что ожидали. Вместо этого вы снова получите сообщение об ошибке, показанное на рис. 13.1 и свидетельствующее о том, что вы не смогли зарегистрироваться. Что же все-таки происходит?

Может даже появиться желание просмотреть код сценария `signin.php`. Имеет ли эта ошибка отношение к сессиям? Похоже, что да. Но она возникает в сценарии `show_user.php`, а не в сценарии `signin.php`. В действительности проблема кроется в функции `authorize_user`, которая находится в сценарии `authorize.php`. Эта функция вызывается в самом начале сценария `show_user.php`:

```
<?php

require '../scripts/authorize.php';
require '../scripts/database_connection.php';
require '../scripts/view.php';

// Авторизация любого зарегистрировавшегося пользователя
authorize_user();
```

Но в этом есть вполне определенный смысл. Функция authorize_user ожидает найти идентификатор пользователя в переменной $_COOKIE, которая выставляется через переменную $_REQUEST.

```
<?php

require_once 'database_connection.php';
require_once 'app_config.php';

function authorize_user($groups = NULL) {

  // Если не создан cookie-файл, проверять группы не нужно
  if ((!isset($_COOKIE['user_id'])) || (!strlen($_COOKIE['user_id']) > 0)) {
    header('Location: signin.php?' .
      'error_message=Для просмотра этой страницы нужно зарегистрироваться.');
    exit();
  }

// и т. д...
```

Но теперь здесь нужно внести еще одно небольшое изменение. Теперь от $_COOKIE нужно перейти к $_SESSION:

```
<?php

require_once 'database_connection.php';
require_once 'app_config.php';

function authorize_user($groups = NULL) {

  // Если не создана информация в сессии, проверять группы не нужно
  if ((!isset($_SESSION['user_id'])) || (!strlen($_SESSION['user_id']) > 0)) {
    header('Location: signin.php?' .
     'error_message= Для просмотра этой страницы нужно зарегистрироваться.');
    exit();
  }

// и т. д...
```

Не забудьте позже внести такие же изменения в функцию при создании строки запроса, используемого для поиска в группах:

```
  // Создание строки запроса
  $query_string =
      "SELECT ug.user_id" .
      "  FROM user_groups ug, groups g" .
      " WHERE g.name = '%s'" .
      "   AND g.id = ug.group_id" .
      "   AND ug.user_id = " . mysql_real_escape_string($_SESSION['user_id']);
```

Выглядит неплохо. К сожалению, вы снова получите не тот результат, который хотели. Зарегистрируйтесь еще раз, и вы увидите еще одно сообщение об ошибке (рис. 13.6). А теперь что происходит?

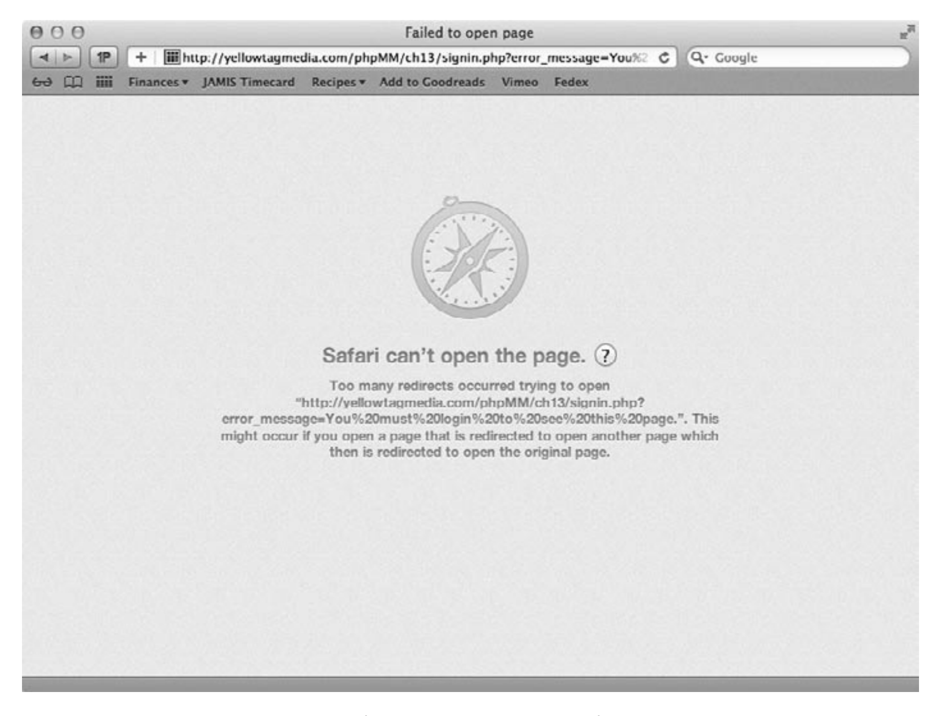


Рис. 13.6. Вы заменили переменную $_COOKIE переменной $_SESSION, но проблема не исчезла

ПРИМЕЧАНИЕ

В зависимости от используемого браузера реакция может быть разной. Вы можете увидеть сообщение о превышении времени ожидания или браузер может просто зависнуть. Во всех этих случаях нет ничего хорошего.

Это произошло из-за неудачного названия функции `session_start`. Оно звучит так, будто функция запускает новую сессию. В таком случае ее можно вызвать всего один, как вы, собственно, и сделали в сценарии `signin.php`. Но каждый PHP-сценарий запускается сам по себе, без связи с какими-либо другими сценариями. Поэтому при вызове сценария `show_user.php` ему ничего не известно о том, что сессия была ранее запущена в сценарии `signin.php`.

Фактически между этими двумя сценариями нет вообще никакой связи, они представляют собой просто два вызова из браузера к чему-то подключенному к Интернету через Wi-Fi или кабель Ethernet. Тогда как два сценария или весь ценный набор сценариев приложения совместно используют эти данные сессии? Ответ будет несколько неожиданным: при вызове функции `start_session` на клиентской машине создается cookie-файл. Да, мы вернулись к использованию cookie-файлов!

Но этот cookie-файл хранит довольно-таки загадочное значение (рис. 13.7). Это значение ссылается на то место, где на сервере хранятся конкретные данные пользователя. Это способ сказать: «Ищите этот код в данных сессии на сервере. Источник там, где этот код».

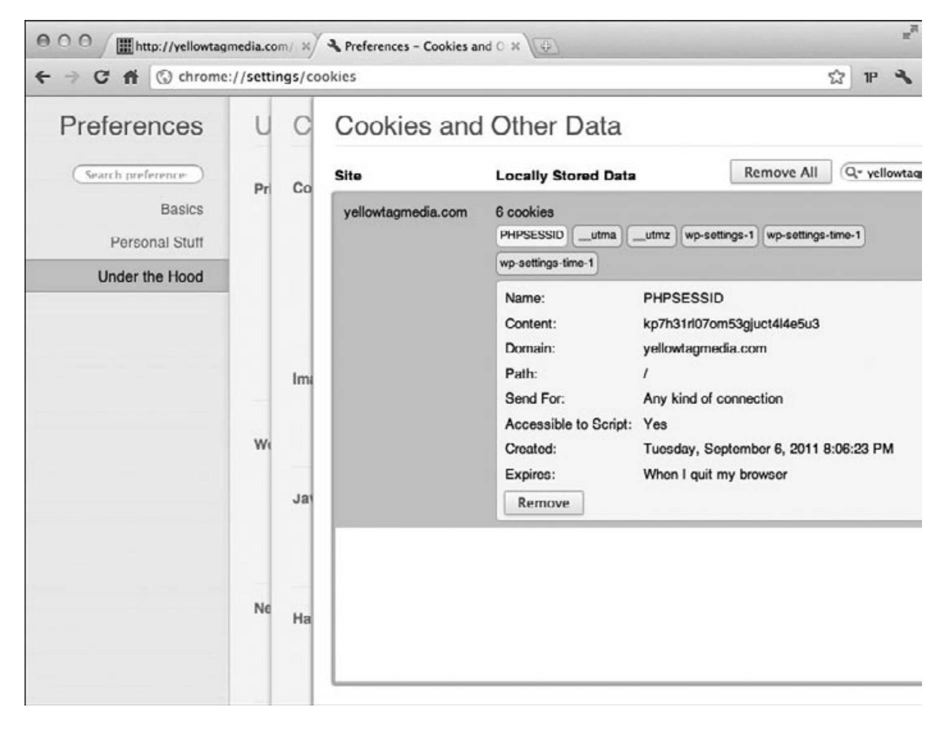


Рис. 13.7. Без cookie-файлов не обойтись, но вам все же удается при этом избежать хранения на машине клиента какой-либо ценной информации

Соответственно, функция выполняет намного больше задач, чем запуск минувшей сессии. Она ищет пользовательский cookie-файл и, если таковой имеется, перезапускает сессию, на которую ссылается данный идентификатор. Следовательно, каждый сценарий, желающий воспользоваться переменной `$_SESSION`, должен вызвать функцию `session_start`.

Это означает, что решение проблемы в сценарии `show_user.php` включает два действия. Первым делом нужно вызвать функцию `session_start` в сценарии `authorize.php`, чтобы гарантировать доступность данных сессии в функции `authorize_user` и в других функциях сценария `authorize.php`.

```
<?php

require_once 'database_connection.php';
require_once 'app_config.php';

session_start();

function authorize_user($groups = NULL) {
  // и т. д...
}
?>
```

Испытайте этот сценарий в работе, и вы увидите сообщение об ошибке, указывающее на второе необходимое действие. Это сообщение, показанное на рис. 13.8, вам уже знакомо.

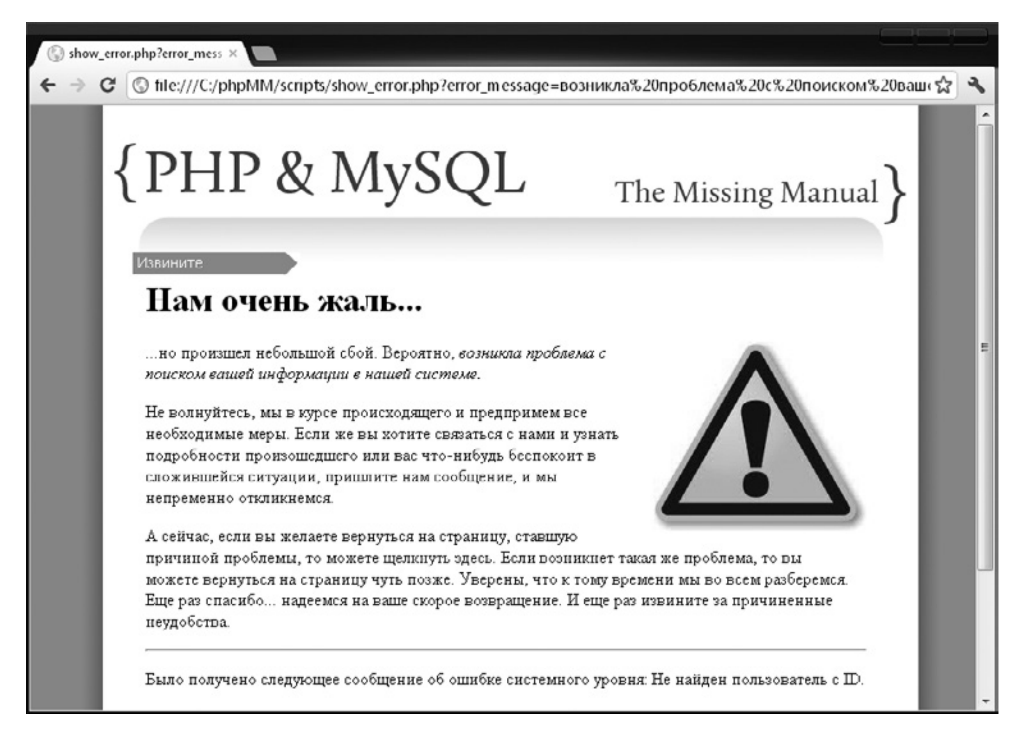


Рис. 13.8. Сообщение об ошибке

Вы уже видели это окно несколько раз. Что же произошло в данном случае? По какой-то причине код, который ищет идентификатор пользователя, не работает и отправляет пользователя обратно с сообщением, что его информация не была найдена.

Переменная $_REQUEST не включает в себя данные переменной $_SESSION

Вот строка в сценарии show_user.php, являющаяся источником проблемы:

```
// Получение ID пользователя, чью информацию нужно показать
$user_id = $_REQUEST['user_id'];
```

Этот код работал, потому что было безразлично, где находится идентификатор пользователя: в $_REQUEST, $_GET, $_POST или $_COOKIE. Все данные из этих переменных появлялись в переменной $_REQUEST. Но теперь идентификатор пользователя пере-

дан другой сверхглобальной переменной, значение которой не включено в переменную $_REQUEST, — в переменную $_SESSION.

Мало того, в сценарии show_users.php до сих пор присутствует код, передающий идентификатор пользователя в параметр запроса:

```
$user_row = sprintf(
  "<li><a href='show_user.php?user_id=%d'>%s %s</a> " .
  "(<a href='mailto:%s'>%s</a>) " .
  "<a href='javascript:delete_user(%d);'><img " .
    "class='delete_user' src='../images/delete.png' " .
    "width='15' /></a></li>",
  $user['user_id'], $user['first_name'], $user['last_name'],
  $user['email'], $user['email'], $user['user_id']);
echo $user_row;
```

ПРИМЕЧАНИЕ

Этот код запрятан где-то в середине сценария show_users.php. Поищите его в цикле while, содержащем HTML.

Следовательно, вы не можете просто переключиться с $_REQUEST на $_SESSION и посчитать, что дело сделано. Вместо этого для всего, что задумано, нужно проверить как $_SESSION, так и $_REQUEST:

```
<?php

require '../scripts/authorize.php';
require '../scripts/database_connection.php';
require '../scripts/view.php';

// Авторизация любого зарегистрировавшегося пользователя
authorize_user();

// Получение ID пользователя, чью информацию нужно показать
$user_id = $_REQUEST['user_id'];

if (!isset($user_id)) {
  $user_id = $_SESSION['user_id'];
}

// Поиск пользователя с помощью $user_id
```

Теперь, если идентификатор пользователя не найден в переменной $_REQUEST, проверяется переменная $_SESSION. А затем перед любой работой с сессией не менее важным действием будет вызов функции session_start:

```
<?php

require '../scripts/authorize.php';
```

```
require '../scripts/database_connection.php';
require '../scripts/view.php';

session_start();

// Авторизация любого зарегистрировавшегося пользователя
authorize_user();

// Получение ID пользователя, чью информацию нужно показать
$user_id = $_REQUEST['user_id'];

if (!isset($user_id)) {
  $user_id = $_SESSION['user_id'];
}

// Поиск пользователя с помощью $user_id
```

Теперь, наконец-то, можно вернуться к просмотру пользовательских профилей.

ПРИМЕЧАНИЕ

В ходе выполнения сценария show_user.php функция session_start вызывается дважды: один раз в сценарии authorize.php, который вставляется с помощью инструкции require_once, а затем в теле самого сценария show_user.php.

Но этот дополнительный вызов не приводит ни к чему, кроме того что заставляет PHP выдать замечание, и нет никакой гарантии, что другие сценарии, которые вставляют в себя сценарий authorize.php, также будут вызывать функцию session_start. Следовательно, продублированный вызов, как в show_user.php, будет случаться не всегда. Лучше считать каждый сценарий самостоятельным. Функцию session_start нужно использовать при каждой работе с сессиями, даже если она может быть вызвана где-нибудь еще.

Меню для любого пользователя?

Теперь осталось только разобраться с меню. В нем по-прежнему используется переменная $_COOKIE, но вы уже точно знаете, что нужно делать. Сначала добавьте крайне важный вызов функции session_start:

```
<?php

require_once 'app_config.php';
require_once 'authorize.php';

define("SUCCESS_MESSAGE", "success");
define("ERROR_MESSAGE", "error");

session_start();

// Далее идут все остальные функции...

?>
```

Затем замените в display_title переменную $_COOKIE переменной $_SESSION:

```
function display_title($title, $success_message = NULL, $error_message = NULL)
{
echo <<<EOD
 <body>
<div id="page_start">
  <div id="header"><h1>PHP & MySQL: The Missing Manual</h1></div>
  <div id="example">$title</div>
  <div id="menu">
    <ul>
      <li><a href="index.html">Главная страница</a></li>
EOD;
  if (isset($_SESSION['user_id'])) {
    if (user_in_group($_COOKIE['user_id'], "Administrators")) {
      echo "<li><a href='show_users.php'>Управление</a></li>";
    }
    echo "<li><a href='show_user.php'>Профиль</a></li>";
    echo "<li><a href='signout.php'>Отмена регистрации</a></li>";
  } else {
    echo "<li><a href='signin.php'>Регистрация</a></li>";
  }
echo <<<EOD
    </ul>
  </div>
EOD;
  display_messages($success_message, $error_message);
  echo "</div> <!-- тег, закрывающий контейнер с id="page_start -->";
}
```

Убедитесь, что все исправлено, и проверьте работу меню. После регистрации вы должны увидеть пункты Отменить регистрацию и Профиль. А когда вы не зарегистрированы, эти пункты не должны быть видны.

А теперь отмените регистрацию

Вопрос об отмене регистрации придется решать заново. При использовании cookie-файлов вы устанавливали время истечения срока их действия на какой-нибудь уже прошедший момент. Работая с переменной $_SESSION, нужно в отношении переменной сессии вызвать функцию unset.

И при всей кажущейся странности вы не можете работать с переменной $_SESSION, не вызвав функцию session_start (даже если эта работа касается сброса значений с помощью вызова функции unset). Поэтому сценарий signout.php должен приобрести следующий вид:

```
<?php

session_start();

unset($_SESSION['user_id']);
```

```
unset($_SESSION['username']);

header('Location: signin.php');
exit();
?>
```

Теперь работа с cookie-файлами ушла в прошлое, и как только будет запущен сценарий `signout.php`, появятся переменные сессии вашего пользователя.

Таким образом, изменив менее 20 строк кода, вы перешли от работы с cookie-файлами к использованию сессий. Неплохой результат. Озабоченные вопросами безопасности пользователи поблагодарят вас за это.

А вы не забыли о проблеме фишинга?

Осталось разобраться еще с одним неприятным явлением. Помните, в главе 7 шла речь о так называемом фишинге (в подразделе «А теперь вас ждут проблемы безопасности и фишинга» раздела «Поиск компромисса для страниц ошибок с помощью PHP»)? Вам пришлось справляться с этой проблемой при использовании переменной `error_message` в качестве параметра запроса к сценарию `show_error.php`. Этот сценарий принимал отображаемое им сообщение об ошибке из параметра запроса:

```
if (isset($_REQUEST['error_message'])) {
  $error_message = preg_replace("/\\\\/", '', $_REQUEST['error_message']);
} else {
  $error_message = "вы здесь оказались из-за сбоя в работе программы.";
}
```

ПРИМЕЧАНИЕ

Этот код находится в файле scripts/show_error.php.

И вы видели, что URL такого содержания:

http://yellowtagmedia.com/phpMM/ch07/show_error.php?error_message=
%3Ca%20href=%22http://www.syfy.com/beinghuman%22%3Eщелкните%20здесь,
%20чтобы%20получить%20описание%20ошибки%3C/a%3E

может создать страницу, похожую на показанную на рис. 13.9. Кажется, здесь нет ничего опасного, но на самом деле это не так.

Но теперь, работая с сессиями, мириться с подобной угрозой безопасности уже не нужно. Проблема заключалась в том, что в параметр запроса попадало сообщение об ошибке. Но теперь, имея в своем распоряжении сессии, вы можете убрать эти сообщения об ошибках из представления. Это означает, что взломщик не получит возможности внедрить в запрос свой коварный параметр, поскольку вы больше не будете использовать такие параметры для этой цели.



Рис. 13.9. Из-за фишинга при CSS-оформлении, соответствующем вашему сайту и форме для приема информации, ваши пользователи могут оказаться обворованными

Вернитесь к файлу scripts/app_config.php и посмотрите на функцию handle_error:

```
function handle_error($user_error_message, $system_error_message) {
  header("Location: " . get_web_path(SITE_ROOT) .
         "scripts/show_error.php?" .
         "error_message={$user_error_message}&" .
         "system_error_message={$system_error_message}");
  header("Location: " . get_web_path(SITE_ROOT) . "scripts/show_error.php");
  exit();
}
```

Этот код превращает текст предоставленного PHP-интерпретатором сообщения об ошибке в параметр запроса. Но теперь это все можно переделать под использование сессий:

```
function handle_error($user_error_message, $system_error_message) {
  session_start();
```

```
  $_SESSION['error_message'] = $user_error_message;
  $_SESSION['system_error_message'] = $system_error_message;
  header("Location: " . get_web_path(SITE_ROOT) . "scripts/show_error.php");
  exit();
}
```

Изменение совершенно простое. И оно делает работу функции handle_error намного понятнее.

Теперь откройте файл show_error.php и внесите соответствующие изменения для извлечения значений из сессии:

```
<?php
  require 'app_config.php';

  session_start();

  if (isset($_SESSION['error_message'])) {
    $error_message = preg_replace("/\\\\/", '', $_SESSION['error_message']);
  } else {
    $error_message = "вы здесь оказались из-за сбоя в работе программы.";
  }

  if (isset($_SESSION['system_error_message'])) {
    $system_error_message = preg_replace("/\\\\/", '',
                                          $_SESSION['system_error_message']);
  } else {
   $system_error_message ="Сообщения об ошибке на уровне системы выдано не было.";
  }
?>
```

ПРИМЕЧАНИЕ

Весь код HTML, следующий за кодом PHP, остался без изменений.

Теперь нужно обновить проблемный URL, чтобы в нем отражалось новое местонахождение файла show_error.php (в вашем каталоге scripts/). Этот URL может приобрести следующий вид:

http://www.yellowtagmedia.com/phpMM/scripts/show_error.php?error_message=
%3Ca%20href=%22http://www.syfy.com/beinghuman%22%3Eщелкните%20здесь,
%20чтобы%20получить%20описание%20ошибки%3C/a%3E

ПРИМЕЧАНИЕ

Вы можете заменить имя домена и обновить путь, но имя файла и параметры запроса должны остаться такими же.

Теперь зайдите на эту страницу в своем браузере. Вы увидите ответ, похожий на тот, который показан на рис. 13.10.

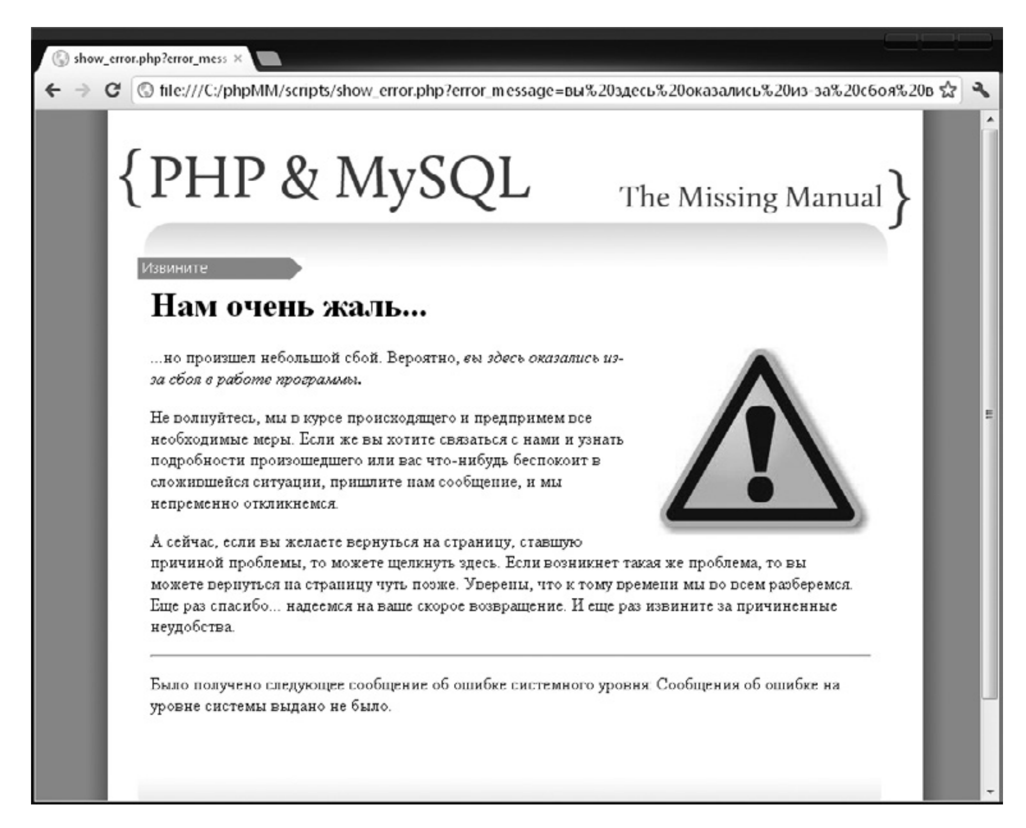


Рис. 13.10. Сессии защищают вас и во многих случаях упрощают ваш код

Теперь угроза фишинга в сообщении миновала. Поскольку сообщение об ошибке хранится в сессии, оно защищено от постороннего вмешательства в это сообщение через URL. Это весьма незначительное изменение имеет огромное значение для ваших пользователей.

А зачем вообще использовать cookie-файлы?

Невольно можно прийти к выводу, что сессии являются ответом на все вопросы. Но это далеко не так. Возможно, самым серьезным ограничением для сессий является то, что с закрытием браузера завершается и работа сессии. Способа обхода этого ограничения не существует. Следовательно, если вы хотите предложить пользователям возможность сохранять регистрацию и при закрытии браузера, сессии для этого просто не подойдут и вам придется применять cookie-файлы.

Кроме того, возможность использования cookie-файлов вам во вред еще не означает, что именно так они и будут применяться. Вы можете задавать истечение

срока использования своих cookie-файлов намного чаще. Вы можете хранить в них только очень небольшие объемы информации и не помещать в них слишком ценные данные. Чтобы избежать хранения наиболее важной информации на машинах пользователей, можно выбрать путь дополнительных поисков в базе данных, даже при условии незначительного замедления работы приложения.

Разумеется, как и практически во всем остальном, вам придется принимать решение, наиболее подходящее для вашего приложения. Но теперь для вас это не составит проблемы. Сейчас вы уже отдаете отчет своим действиям и знаете, каким арсеналом при этом располагаете. Разумно используйте все это, экспериментируйте и учитесь... постоянно учитесь.

Маклафлин Б.

PHP и MySQL. Исчерпывающее руководство

Перевел с английского Н. Вильчинский

Заведующая редакцией	*К. Галицкая*
Руководитель проекта	*Д. Виницкий*
Ведущий редактор	*Е. Каляева*
Художник	*Л. Адуевская*
Корректор	*О. Андросик*
Верстка	*Г. Блинов*

ООО «Мир книг», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.

Подписано в печать 24.07.12. Формат 70×100/16. Усл. п. л. 41,280. Тираж 2000. Заказ 0000.

Отпечатано с готовых диапозитивов в ГППО «Псковская областная типография».
180004, Псков, ул. Ротная, 34.